DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

VLSI TUTORIALS THROUGH THE
VIDEO-COMPUTER COURSEWARE
IMPLEMENTATION SYSTEM

by

Liesel R. Muth

March 1985

Thesis Advisor: D. E. Kirk

Approved for public release; distribution is unlimited.

T223868

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)*: VLSI Tutorials Through the Video-computer Courseware Implementation System	5. TYPE OF REPORT & PERIOD COVERED: Master's Thesis; March 1985
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Liesel R. Muth	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, CA 93943	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Postgraduate School, Monterey, CA 93943	12. REPORT DATE: March 1985
	13. NUMBER OF PAGES: 149
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*	15. SECURITY CLASS. *(of this report)*: UNCLASSIFIED
	15a. DECLASSIFICATION/ DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*
Approved for public release; distribution is unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*
CAD; Computer-Aided Design(CAD); CAI; Computer Aided Instruction (CAI); UNIX; VLSI Design; VCIS; Video-Computer Courseware Implementation System (VCIS)

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*
An overview of the University of Utah Video-computer Courseware Implementation System (VCIS) programs is presented. A tutorial is included which is designed to provide an author with enough knowledge to create a simple lesson containing test and graphics frames. VCIS was used in creating two tutorials, which exercise most of the functions in the authoring system. One tutorial is only text and is an introduction to the Berkeley UNIX operating system and its vi editor. The other tutorial contains both text and graphics and is an introduction to Very Large Scale Integrated (VLSI) circuit design. An evaluation of the VCIS is also included.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

Approved for public release; distribution is unlimited.

VLSI Tutorials Through The Video-computer Courseware Implementation System

by

Liesel R. Muth
Lieutenant, United States Navy
B.A., The University of Michigan, 1976

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 1985

ABSTRACT

An overview of the University of Utah Video-computer Courseware Implementation System (VCIS) programs is presented. A tutorial is included which is designed to provide an author with enough knowledge to create a simple lesson containing text and graphics frames. VCIS was used in creating two tutorials, which exercise most of the functions in the authoring system. One tutorial is only text and is an introduction to the Berkeley UNIX operating system and its vi editor. The other tutorial contains both text and graphics and is an introduction to Very Large Scale Integrated (VLSI) circuit design. An evaluation of the VCIS is also included.

TABLE OF CONTENTS

I. INTRODUCTION 9

II. THE VIDEO-COMPUTER COURSEWARE IMPLEMENTATION SYSTEM (VCIS) 11

A. HARDWARE 11

B. PROGRAMS 11

1. Introduction 11

2. TEXTEDIT 12

3. GRAFEDIT 13

4. BUILDER 14

5. INTERP 15

6. MANAGER 15

7. Other Programs 16

C. CONCLUSIONS 17

III. USING VCIS 18

A. INTRODUCTION 18

B. LESSON PLANNING 18

C. SETTING UP AND PROGRAM CONVENTIONS 19

D. TEXTEDIT 20

1. Initial Steps 20

2. Creating A Frame 21

3. Changing Text 28

E. GRAFEDIT 29

1. Initial Steps 29

2. Creating a Frame 30

3. Creating an Object 32

4. Making a Frame 36

5. Viewing Other Frames 38

6. Changing Frames And Objects 40
F. BUILDER 42
1. Structure 42
2. Initial Steps 42
3. Building a Lesson 45
4. Editing a Segment 52
G. INTERP - TEST MODE 53
1. Initial Steps 53
2. Segment Testing 53
H. LINKER 54
1. Initial Steps 54
2. Linking Segments 54
3. Possible Errors 55
I. MANAGER 55
J. INTERP - FINAL MODE 57
K. LISTFRAM 57
L. SUMMARY 59

IV. CREATION OF VLSI TUTORIALS 60
A. INTRODUCTION 60
B. LESSON PLANNING 60
1. Structure 60
2. Questions 61
3. Assistance 62
C. TEXT 62
1. Text Composition 62
2. File Order 63
3. Attribute Selection 63
4. Text Input 64
5. Text Modifications 64
D. GRAPHICS 64
1. Object Selection 64
2. Object and Frame Creation 64
3. Object and Frame Modification 65

E. BUILDING THE LESSON 66
1. Using BUILDER 66
2. Final Lesson Made with MANAGER and INTERP 68
F. STUDENT COMMENTS 68

V. EVALUATION OF VCIS 69
A. INTRODUCTION 69
B. GRAFEDIT 69
C. VCIS AVAILABILITY 69
D. COLOR SPECTRUM CONFLICTS 70
E. LINKER 71
F. NAMING CONVENTIONS 72
G. MANAGER 72
H. OVERALL COMMENTS 73

APPENDIX A: UNIX TUTORIAL 74

APPENDIX B: SELECTED FRAMES FROM THE VLSI TUTORIAL . . 135

APPENDIX C: SYSTEM REQUIREMENTS AND LOGON PROCEDURE FOR VCIS 140
1. Diskettes 140
2. Logon Procedure 140
3. Logging Off 141
4. Executing The VIUNIX Tutorial 141

APPENDIX D: SURVEY ON TUTORIAL 143

LIST OF REFERENCES 147

BIBLIOGRAPHY 148

INITIAL DISTRIBUTION LIST 149

LIST OF FIGURES

3.1 TEXTEDIT Status Screen 1 23
3.2 TEXTEDIT Status Screen 2 24
3.3 GRAFEDIT Status Frame 31
3.4 Sample Frame 33
3.5 Examples of Path Terminators 43
3.6 Menu Segment 44
3.7 Called Segment 45
3.8 BUILDER Status Board 46
3.9 Manager Status Board 56
B.1 VLSI Tutorial Logo 135
B.2 Summary of Basic Shapes, Colors, and Combinations 136
B.3 Butting Contact 137
B.4 Review 138
B.5 Mixed Notation 139

ACKNOWLEDGEMENTS

I would like to thank Barbara Knapp and all the other University of Utah programmers. Their cheerful help was of immense assistance to me in using the VCIS to create the tutorials. I would also like to thank my friends, especially LT M. Mort, for their support during the writing of my thesis.

I. INTRODUCTION

The potential of computer-aided instruction (CAI) in today's educational environment is great and is growing steadily. With the accessibility of personal computers, the use of computer-aided instruction becomes an even more attractive option for the instructor.

In the area of VLSI design, the use of computer-aided instruction is particularly worthwhile due to the increasing use of computer-aided design (CAD) tools. The material in the lessons themselves is ideally suited to the CAI environment, because in VLSI design part of the initial familiarization process is recognizing the shapes and colors as they are presented on a computer monitor screen. Also, use of these tools requires a considerable amount of orientation before the student is able to use them for design. The classroom environment is not ideal for teaching use of CAD tools. This orientation would be best, of course, if it could be done individually with one-on-one instruction involving the professor and the student, but even with small classes of students, this ideal situation is rarely possible. A well constructed lesson using CAI can be nearly as effective as individual instruction. It allows the student to proceed at his/her own pace and at a time that is convenient for him/her. It frees the professor to answer more questions from the rest of the students.

Attractive though this option is, there can be difficulties in developing a lesson that will function in this environment. In any environment, of course, the instructor must be able to present the lesson material in a clear and concise format, but in a computer environment, this becomes even more critical. Not only is the instructor generally

not present to answer questions on the lesson material, but the man-machine interface can add to student frustration.

The lesson or tutorial must be both useful to the student, and possible for the professor to develop in a reasonable amount of time. Rather than attempting to combine text-editing programs and graphics production programs, and then finally having to develop a program to control the lesson structure, there are CAI authoring systems to lessen the arduous task of lesson production without use of such a system.

The interest in CAI has been growing at the Naval Postgraduate School, for use in explaining CAD tools, as well as in other areas. After considering various authoring systems, in March, 1984 NPS received copies of the Video/computer Courseware Implementation System (VCIS) produced by the University of Utah Research Foundation. VCIS possesses programs which enable the author to produce text and graphics frames, and select videodisk or videotape frames. VCIS also provides the programs to build the lesson from these component parts. Once finished with constructing the lesson, the authoring system can be used to test, execute, and print the tutorial. Two tutorials, which together exercise most of the functions of the authoring system, were created using the VCIS.

An overview of the VCIS programs currently available is provided in Chapter II. Chapter III is a tutorial on the VCIS, which is intended to enable an author to create a simple lesson. Chapter IV describes the two tutorials produced using the VCIS and Chapter V evaluates the VCIS.

II. THE VIDEO-COMPUTER COURSEWARE IMPLEMENTATION SYSTEM (VCIS)

A. HARDWARE

The University of Utah Video-computer Courseware Implementation System can be used on a variety of micro- or mini-computers under several operating systems. The micro-computers currently supported are Sirius, Terak 8510a, Zenith Z-100, IBM PC or XT and true IBM compatibles, Sperry Univac micro, and HP 9836. Current efforts are underway involving the Tandy 2000 and the MacIntosh. The VAX 11-750 and the VAX 11-780 are also supported. VCIS runs under four operating systems at present: Microsoft DOS Version 2.0 and later, Berkeley UNIX 4.2, UCSD Pascal, and HP Pascal.

Each one of the microcomputers listed in the previous section needs 256K of memory to run VCIS. The combination of a hard disk and one or more disk drives makes file manipulation easier but is not necessary.

An actual VCIS workstation consists of a microcomputer, or a graphics terminal, connected to a minicomputer with a videodisc or videotape player under computer control. The station has either a 9" or 13" black and white or color monitor or TV. If the microcomputer is an IBM PC, its color capabilities may be augmented by use of the Tecmar Graphic Master color board, or the regular IBM board may be used.

B. PROGRAMS

1. Introduction

The programs are as described in VCIS User's Guide 2.0d and 3.0a. See [Refs. 1,2] for complete descriptions of

the commands. The following sections provide a brief overview of the programs and use capitalized words to indicate use of a VCIS command.

2. TEXTEDIT

a. Capabilities

TEXTEDIT is the program used to create the text portion of a lesson. Lesson text can be composed of up to 254 segments with up to 254 frames per segment. A text frame is a grid of 23 (vertical) by 80 (horizontal) character positions. The 24th line of the frame is the command line. A frame is comparable to a transparency which may be overlaid with another text or graphics frame. If the frames are properly constructed, the underlying frame or frames will remain visible.

The number of colors available for background and foreground depends on the system. Eight background colors and 16 foreground colors are supported. On systems without color, the background color is usually ignored and the foreground colors are sometimes displayed with varying intensities. The background color is not saved as part of the text file, however, and has to be re-specified later in BUILDER.

There are two complete 96 character sets available with most systems (one with and one without underlines). Another program, CHEDIT, provides the capability to create additional character sets (one using Greek letters, for example). There are 255 possible text attributes (type of character, color of foreground, color of background) from which to form the working group used for a lesson.

b. Text Entering or Changing

Text is entered initially with the INSERT command. When the insertion is complete, it is <ACCEPT>ed or removed if undesirable with <ESC>. After text has been entered, it can be MOVEd or DELETEd or changed to a NEW color or character set. As the text is entered or changed, the author must remember the difference between a space (using the spacebar) and a null-space (using the cursor). Frames are like transparencies and a space appears as a bit of background color. Several frames can be merged together to create a new frame, again using the concept of frame transparency.

At any time during the text creation, the author can VIEW any or all earlier frames or any graphics frame. This feature makes it easy for the author to see how the text and graphics frames combine together, as well as how the text alone is progressing.

3. GRAFEDIT

a. Capabilities

GRAFEDIT is the program used to create and modify the graphics segments of lessons. As with TEXTEDIT, GRAFEDIT is divided into 254 segments, where each segment may contain up to 254 frames. The actual size of the graphics frame depends on the resolution of the individual monitor.

Eight to 16 colors are available depending upon the system used. A different color may be selected for each of several options - ZONE (fill large area), LINE, FILL (fill small area), MARKER or BACKGROUND. These selections become the working set. The working set may be temporarily altered at any time and will revert to the original working set, following the completion of that particular command.

There are up to nine pattern indices available. They range from solid through diagonal, vertical and horizontal lines to dots to a blinking pattern. The programmer has the option to change the kind of pattern available. The size of the elements within the pattern does not change as an object is increased or decreased in size.

b. Object Creation/Modification

The basic unit of the graphics frame is the object. It can be created using one of the seven predefined graphics objects (line, circle, arc, bezier curve, marker, fill or zone) or using other author defined objects. The object is created by moving the cursor to any point on the screen using a digitizer, keyboard or mouse. An object consists of groups of control points (a circle has two control points - the center and a point on the circle). A grouping starts when the first control point is entered and it ends when another option (LINE, ZONE, etc.) is chosen. The object can be modified at any time to any degree. The author also can use objects created on other frames, segments or files to create a new frame or object. As with TEXTEDIT, the author can overlay the current graphics frame with any text frame.

4. BUILDER

BUILDER creates the actual lesson structure. This program orders the presentation of the text, graphics, and video frames previously created. It sets up the question patterns - what answers are allowed or anticipated (right or wrong), in addition to responses for unanticipated answers.

BUILDER segments are composed of video, graphics and text frames and may be of any length, subject only to the amount of available room on the disk. A segment may call or USE other segments. It may also call a large "special" (up

to 10K words of code) or a small "special" (up to 6K words of code). "Specials" are Pascal programs written to add features not supported by VCIS. A special would be used to create a graph from a student's input, for example.

A lesson segment is created by setting up various paths for a student to follow. Each decision point has various options available to the author. The student may continue directly, branch to a help segment, back up to view previous frames, or quit. The author must have planned all options, as unfinished paths are not allowed.

5. INTERP

The lesson segments can be tested, or used in their final form, by using INTERP. INTERP uses the options specified in MANAGER to decide, for example, if student path and answers are to be recorded, or if backing up is allowed. If other segments are called (USEd) during a test run, INTERP will assume the segment was available and was properly executed, and it will continue. For testing purposes, INTERP requires the four files per segment created by BUILDER (.GRAF, .RUN, .PAG and .TXT). In the testing mode, INTERP provides tracing information on the monitor which shows the segment and question that is currently being used. When the lesson is completed and INTERP is in the final mode, the file MANAGER.DAT must be present. In the testing mode, INTERP uses only those commands written by BUILDER, but other options must be selected for the final run that MANAGER.DAT provides (options like allowing backing up or recording student path).

6. MANAGER

MANAGER is the program that creates the file MANAGER.DAT used by INTERP. It also creates the list of students who are allowed to access the lesson if passwords

are to be used. MANAGER holds the list of right, wrong and neutral replies used by INTERP when a random reply is needed. Student paths and answers are recorded in MANAGER.DAT for future reference.

7. Other Programs

a. LISTFRAM

LISTFRAM lists text or graphics frames. Output may be to the monitor or to a printer. A partial or a complete listing of text or graphics frames is allowed.

b. CHEDIT

CHEDIT creates or edits character sets. A regular size grid (10 dots high by 8 dots wide) and a wide size grid (10 dots high by 16 dots wide) are available. The regular size grid creates a character set of 192 possible characters, while the wide size grid has a set of 96 characters.

c. LINKER

The LINKER programs links one control segment with up to 8 additional segments. If a lesson has more than eight segments, LINKER can be used as many times as needed doing no more than eight new segments at a time. If one segment has been changed, the whole lesson does not need to be relinked. Only the new segment must be relinked to the old lesson, where it will replace the older version.

d. SCOMPARE

SCOMPARE provides the facility of checking possible string combinations as used in BUILDER. This allows the author to see if wrong and right answers will be properly detected when entered by the student.

e. SELECT

SELECT picks the video sequences for a lesson. The author uses SELECT to obtain the start and stop points of the sequence for later use in BUILDER.

C. CONCLUSIONS

With the basic VCIS workstation containing a micro- or mini-computer with a graphics terminal and an overview of the functions available under the authoring system, the next step is to create a CAI lesson at the workstation using the programs. Chapter III provides a tutorial which if used with the VCIS User's Manuals, will assist the author in creating a simple lesson using text and graphics.

III. USING VCIS

A. INTRODUCTION

This chapter is intended to provide a basic knowledge of VCIS. Upon completicn, the reader should be able to create a simple lesson with basic text, standard (system) character set and simple graphics. Explanations for all of the commands for a given program or all the options within a given command are not provided. See [Refs. 1,2] for further details and explanations. The convention used here is that program names are completely capitalized, while command names within a program have only the first letter capitalized and are enclosed in quotation marks. Specific terminal keys are indicated by completely capitalizing the name of the key and by enclosing the name in "<>".

The explanations are given in terms of an IBM PC with a Tecmar Graphic Master color board, however, usage on other computers is similar. While VCIS permits use of the mouse, digitizer, or keyboard for command input, only the keyboard commands are given .

The programs are presented in the sequence that the author would probably use in creating a lesson, except for TEXTEDIT and GRAFEDIT which are interchangeable. The sequence is TEXTEDIT, GRAFEDIT, BUILDER, INTERP, LINKER, MANAGER, and LISTFRAM.

B. LESSON PLANNING

Before attempting to use any of the VCIS programs, the author must determine the lesscn structure. Lesson structure starts with the overall structure - whether the lesson will be menu-driven or will be strictly sequential, for

example. Lesson structure also includes the structure of the frame - if the frame is purely factual, or if the student is asked a question on lesson material.

VCIS allows the author tc ask questions of various types. For example, the author may ask questions which require the student to do a numerical calculation, provide a string response or make a choice among author-supplied responses. In each instance, the author must determine a response to the three types of possible student answers, which are right, wrong or unanticipated. Each anticipated right or wrong answer may have its own response.

The text and graphics on each frame have to be edited to provide a clear and concise presentation of a minimum number of new ideas. Especially when both text and graphics are used, the author must take care to avoid a cluttered and crowded screen. While the lesscn can be structured to allow the student to back-up and review a previous frame, careful editing of each frame will minimize this. A clear presentation with unambiguous questions will minimize the number of wrong and unanticipated answers.

C. SETTING UP AND PROGRAM CONVENTIONS

The system is booted up using the Microsoft DOS V2.0, with a system file that structures the use of the F7, F8, F9, and F10 function keys for use as cursor control keys (up, down, left, and right, respectively). Once booted, all of the programs described (except MANAGER) will return the author to the operating system upon completion. See Appendix C for details on what disks are needed and on the logon procedure.

Except where indicated, the author invokes a command only by typing the first letter (which is capitalized) of the command. No additional ENTER or RETURN keystroke is

required. A command is usually terminated with either the <DEL> (accepts the action of the command) or the <ESC> (usually aborts the action of the command).

D. TEXTEDIT

1. Initial Steps

Before invoking the TEXTEDIT program, the author should be sure that there is sufficient room on the diskette to save the file and its changes. This is done by using the Microsoft Dos command "dir<RET>". If no drive is specified, the default (shown by the prompt "A>" or "B>") is used. The directory of the other drive is checked, without changing drives, by "dir b:<RET>", or "dir a:<RET>". The directory displays files with their sizes, the date the files were last changed, the total amount of space used, and the amount of space remaining on the diskette. TEXTEDIT itself requires 118K bytes of space, while a sample file containing one segment with 68 frames, needs 64K bytes.

TEXTEDIT is invoked by typing TEXTEDIT<RET>. The first level command line (line 24) displays:

```
Create Edit List
```

"Create" sets up a new text file, while "Edit" is used for changing or adding to a previously created text file. "List" is used to list the files on the disk. If selected, "Create" requires a segment name. Segment names should clearly indicate their position and/or function within a lesson. The filename is specified at the end of the session. A new segment starts with frame number 1. "Edit" asks for the filename and when obtained (the file may reside on a disk in the other drive, for example), TEXTEDIT

displays information on the file and asks if the author wants to start a new segment. The information displayed is the number of segments in the file, the number of times a segment has been revised and the size of the segment in frames and blocks. The author decides whether to start a new segment by answering Y (Yes) or N (No). A positive response elicits a request for the new segment name while a negative response causes the system to ask for a current segment name. As in "Create", a new segment starts with frame number 1. A previously created segment starts with a new frame with the next number (for example, if 31 previous frames exist, the next frame is 32).

The command line for "List" displays:

```
All Directory Font Graphics Pattern Text Remove
```

The commands "All", "Font", "Text", and "Graphics" list those files of the types specified that reside on the disk. "Directory" provides the author with the opportunity to change the directory name. "Pattern" attempts to match a pattern string against file names on the disk.

2. Creating A Frame

a. Set Up

Once the new frame appears, the second level command line is:

```
Delete Yank Xchange New -- Adjust Move      ? <.>
Keep Edit Copy Remove View -- Page Graphics ? <.>
Undo Find Status "List" -- Move Tab -- Quit   ? <.>
```

As shown, the second level ccmmand line appears as three lines, and the author cycles from one line to the next by use of the period <.> key. It is not necessary for a desired command to be visible on the command line for it to be invoked. A one line summary of commands is provided with <?>.

TEXTEDIT is initialized with certain default values which are shown in Figure 3.1 and Figure 3.2. The character set, or font, is the one provided by the system, which is similar to most typewriter keyboards (no mathematical symbols, for example). The default attributes are blue, red, purple/pink, light green, light blue, yellow and white foreground colors with black highlighting. The frame background is black.

These default values can be changed at any point during the text editing process. The author can view or change the default values by invoking "Status" (type "S"). The first screen of "Status" (Figure 3.1) has the command line:

```
Segment_name Font Background Edit List Quit
```

"Segment_name" allows the author to change the current segment name. "Font" allows the author to name a new character set other than the standard set which is listed as the default. The character set file must be available, either on the same disk or on the hard disk, or an error will occur. "Background" prompts the author for the frame's background color (0-7) when invoked. The background color is not saved with the text frame and must be respecified later, when the text frame is used in BUILDER. "Edit" displays the total set of 255 attributes available (Figure 3.2). The default values are indicated by asterisks. These

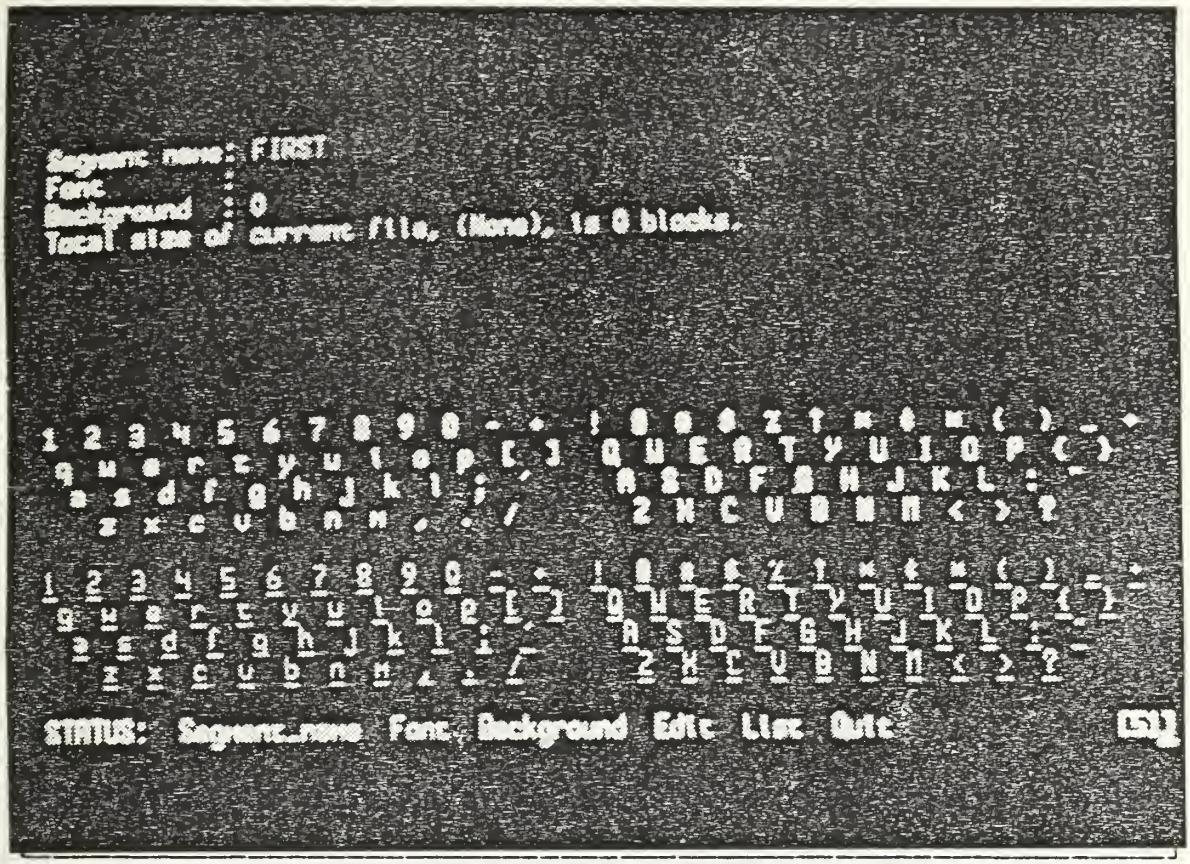


Figure 3.1 TEXTEDIT Status Screen 1

attributes have the normal character set (no underlining) in the upper half of the total set, with the alternate character set (with underlining) in the bottom half. Each attribute is displayed as a number with one of eight background colors and one of 16 foreground colors. "List" displays the same information about the current text file as would be displayed when TEXTEDIT was first entered.

On the second screen of "Status", the cursor is homed on the first attribute in the upper left corner of the attribute matrix. An attribute is selected or removed from the working set by moving the cursor to the desired position with the function keys F7-F10 and depressing <DEL> to accept or remove that attribute. This new working set is not saved

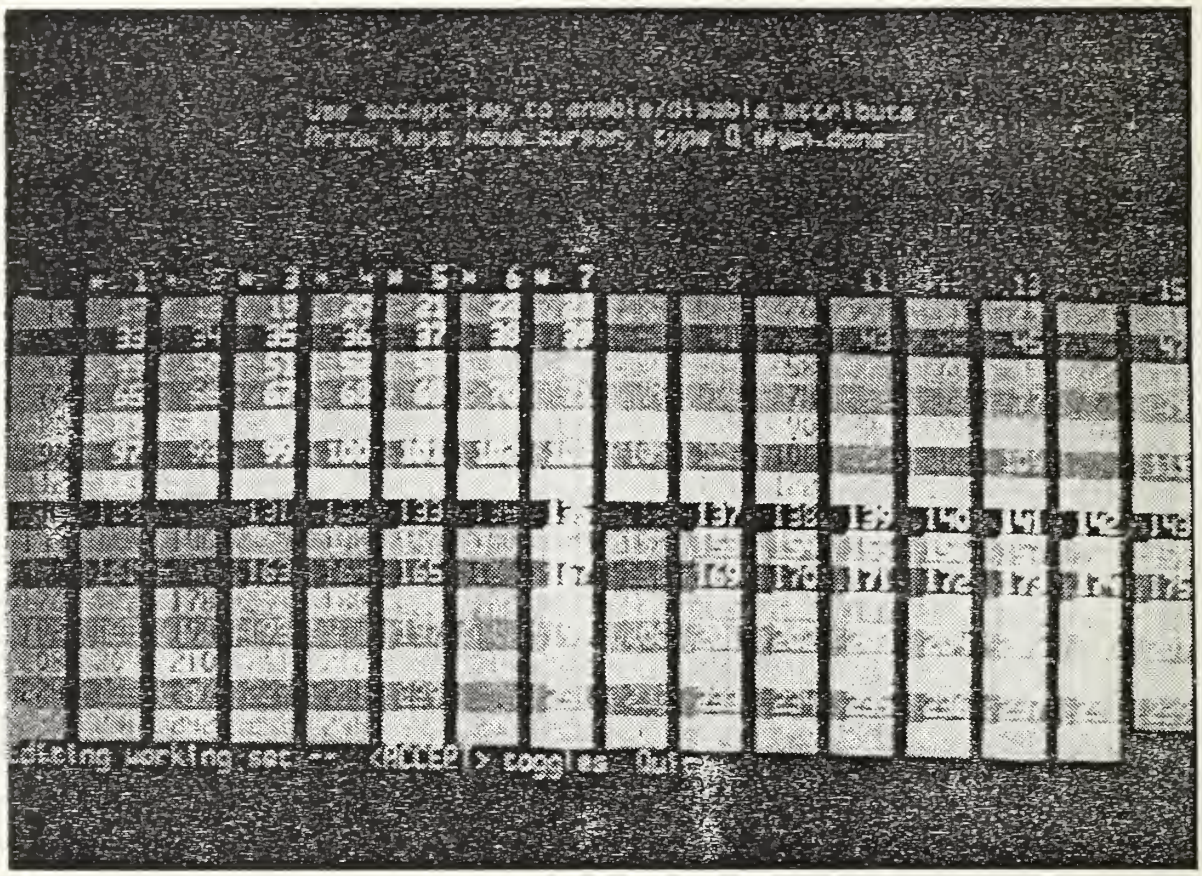


Figure 3.2 TEXTEDIT Status Screen 2

with the text frame, and must be regenerated each time the file is edited. The author stops the attribute editing session with "Quit", which returns to the "Status" screen one, and an additional "Quit" is needed to return to the second level command line.

b. Putting Text on the Screen

The author selects the desired attribute for text input by cycling through the working set by holding down the <CONTROL> and G keys simultaneously. The second level command line then displays the current attribute. Text is placed on the screen by invoking "Insert" and typing in the text. The author must either type <RET> or

re-position the cursor at the end of each line as the text does not wraparound around the screen. There will be a beep (prompt) if the author attempts to put text past the end of a line. The author can put text at different places on the screen by moving about with either the function keys, or a combination of <SPACE BAR> and <RET>.

The effects of the function keys and that of the <SPACE BAR> are entirely different. The function keys produce a null space, while the <SPACE BAR> yields a space made in the background color. This is important only when one frame overlays another, when what appears to be an empty area on the top frame is actually an opaque screen. The best way to avoid any difficulties, is to use the cursor position keys when moving the cursor from one part of the screen tc another. Frequent use of "View" will check on any unwanted opaque areas.

At any point during the insertion, the author may change attributes, by again cycling through the working set using <CONTROL> and G. This only affects future text and not that already placed on the frame. Mistakes can be corrected by either backspacing (erases the text) or positioning the cursor with the function keys, and inserting the new/correct text or characters. The "Insert" command ends with <DEL> (to save) and <ESC> (to destroy) changes.

c. Viewing Other Frames

When text and graphics will be combined later on the same lesson frame, the author must make frequent comparisons between the current text frame and its associated graphics frame or frames. The TEXTEDIT command "Graphics" starts by requesting the file name, then the segment name (only if there is more than one segment). Once the graphics file has been specified, the same file will be used if the "Graphics" command is again invcked, unless a different file

is requested. Once the graphics file and segment names are known, the "Graphics" command line is:

```
Get Segment Clear Frame Next Previous <ACCEPT> <ESC>
```

"Get" changes the graphics file specified when "Graphics" was first invoked. "Segment" changes the segment used in the "Graphics" command. "Clear" erases all graphics frames. "Frame" asks for a frame number and overlays that frame on the current text frame. "Next" and "Previous" overlay the next and preceding frames on the screen, providing that a frame number has already been specified. The specified graphics frames are kept on the screen with the <DEL> key. This also returns the author to the second level command line. Once accepted (<DEL> key), the displayed graphics frame or frames remain with the frame, until a "Remove" command is issued or until a new text frame is displayed (for example, use of "Insert" creates a new text frame). <ESC> also returns the author to the second level command line, but the graphics frame or frames are cleared from the frame.

The author can check for consistency of the position of the material on a line (useful for command lines) or wording of text, or can merge frames by using the "View" command. "View" produces the command line:

```
{segment name} Segment Frame Next Previous Clear <ESC>
<ACCEPT>
```

Initially, the current segment name is displayed on the command line. This may be changed with the command "Segment". "Frame" requests the frame number within the

named segment to be overlayed on the current text frame. "Frame" and "Segment" prompt the author for a number and a name, respectively. "Next" and "Previous" display the next and preceding frames if a frame number has been selected earlier. "Clear" removes the overlayed frame. <ACCEPT> (the <DEL> key) merges the overlaid text frame or frames with the current frame, and returns the author to the second level command line. <ESC> also returns to the second level command line with the viewed frame or frames remaining on the screen, but without merging them with the current frame.

d. Ending the Frame and the File

When a text frame is completed, "Keep" is invoked. It is not possible to progress to the next frame unless the current frame has been kept. When a frame has been kept, TEXTEDIT cycles to the next frame, and the text creation process repeats.

When the author types "Quit", the command line becomes:

```
Save Discard Return
```

"Save" asks for a text file name. The author is prompted with the name of the current text file, if there is one. If the author hits <RET>, signifying that the current text file name is to be used for the changed text file, TEXTEDIT asks if the previous version should be replaced. There are no difficulties with replacing (writing over) the previous file. "Discard" destroys any changes made during the session, and the prompt asks if the author wishes to reconsider the decision to discard. "Return" continues the TEXTEDIT session where it left off.

3. Changing Text

Text frames can be modified at two different times, before and after a given frame has been kept. The actual changing process is the same for both. A section of text can have its attributes changed using the "New" command. Either before or when "New" is invoked, the author cycles the working set of attributes to the one desired, using <CONTROL> and G pressed simultaneously. The new color must be the one used for the command line when the marking procedure of "New" commences. Only one color can be changed at a time. If several new colors must be changed, each is a separate invocation of "New". When "New" asks for the upper left corner (which is under the upper left character or space), the author positions the cursor in the proper position and hits the <DEL> key. The lower left corner is defined in the same manner. After the change is on the screen, the author is asked if the change is acceptable. A <DEL> keeps the change, and <ESC> aborts the change, and both return the author to the second level command line.

"Insert" is used to insert characters. As described earlier, "Insert" does not wrap around, and ends with <DEL> or <ESC>. "Delete" removes the character to the right when the <SPACE BAR> is used, and the rest of the line when <RET> is used. Backspacing restores a deleted character. The deletion process, as with "Insert", ends with <DEL> or <ESC>. "Xchange" exchanges character for character and ends with <DEL> or <ESC>. As with "Delete", backspacing returns the exchanged character to its original value.

To reposition a block of text on the frame, the "Move" command is used. The author is asked to mark the upper left corner, then the lower right corner of the block to be moved. As with "New", the cursor position for each corner is under the respective corner or space. The next

question from "Move" asks for the upper left corner of the destination postion (under the character), again marked with <DEL>. The command's actions can be aborted at any time with <ESC>. If the moved text extends outside the frame's edges, the author is informed and is asked if the move should be done. A positive response destroys the overhanging portion of the text. "Move" is terminated with either <DEL> or <ESC>.

E. GRAFEDIT

1. Initial Steps

Before invoking the GRAFEDIT program, the author should be sure that there is sufficient room on the diskette to save the file. GRAFEDIT itself requires 137K bytes, plus 2K for the NOMOUSE program. For example, a one segment, 37 frame graphics file takes up 47K bytes of space.

GRAFEDIt is invoked by typing GR_IEC<RET>. The first level command line is:

```
Create Edit Quit
```

As in TEXTEDIT, "Create" asks for a segment name, starts in Frame 1 and names the file at the conclusion of the session. Segment names should clearly indicate their postion and/or function within a lesson. Also as in TEXTEDIT, "Edit" asks for the file name, then displays the same sort of information on the file as described under TEXTEDIT. The author then either chooses to start a new segment or picks a previously created segment.

2. Creating a Frame

a. Set Up

Once the new frame appears, the second level command line is:

```
Create Edit Draw Modify Remove Keep Get View Help Stat
X Text Quit.
```

Most of the second level commands are explained later in the order that the author would probably use them - starting with "Status", "Draw", "Create", "Get", "View", "Keep", "Edit", and "Modify".

The remaining two commands are "Help" and "Remove". "Help" displays a frame containing one line descriptions of the second level commands. "Remove" displays the command line:

```
Graphics All Unused Object Ncthing ? Esc
```

"Graphics" removes only what has been drawn on the screen, while "All" deletes both what is on the screen and all objects, used and unused. "Unused" removes those objects not used in what has been drawn on the screen. "Object" displays a numbered list of all the objects on the frame and the author is prompted for the number of the object to be removed from the list. For each of the previously described commands, the author is asked to confirm that the item or items specified are to be deleted. "Nothing" and "Esc" escape "Remove" without deleting anything. "?" is the help command and was not available on version III.0a of GRAFEDIT.

GRAFEDIT is also initialized with certain default values which are shown in Figure 3.3. The author invokes "Status" to view or change the default values. The default values are white solid line, white

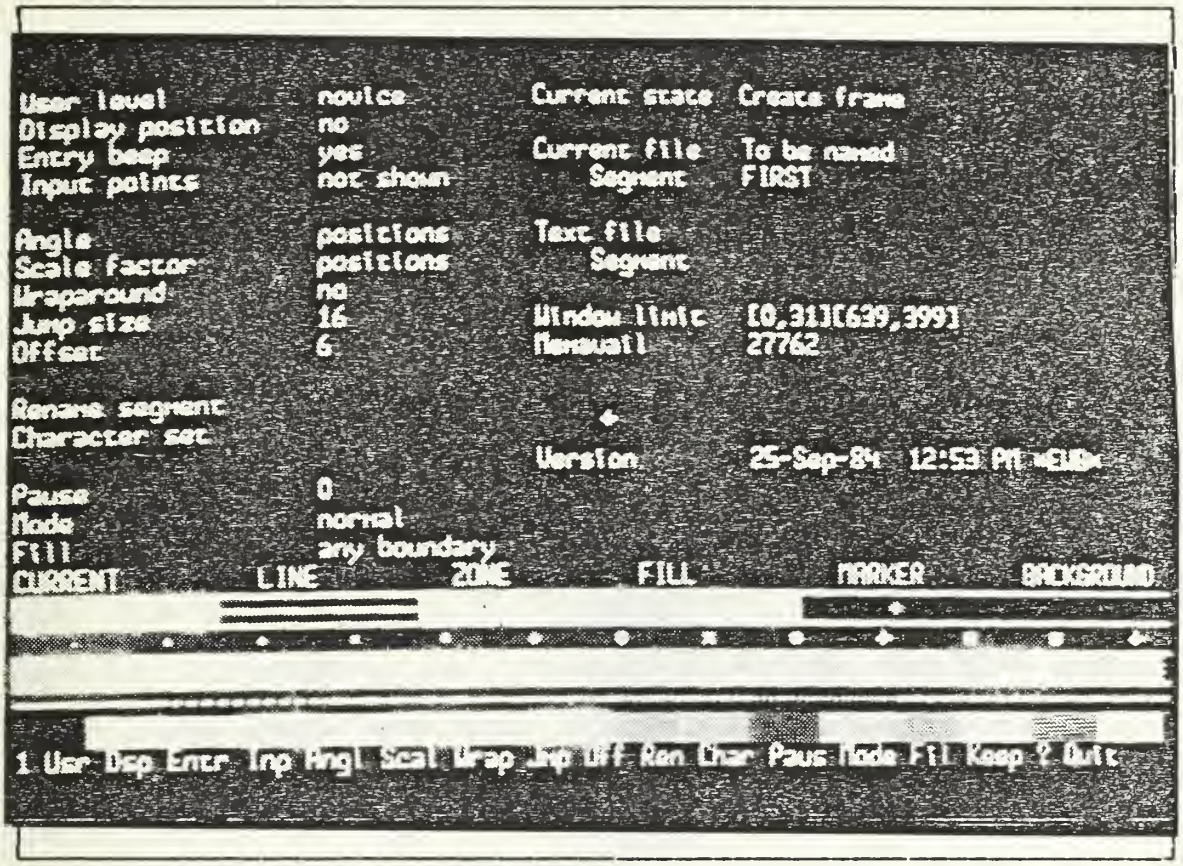


Figure 3.3 GRAFEDIT Status Frame

solid zone and fill, white small dot marker, and black background. Other default values with which the author may be concerned are the arrow skip value ("Ju"), which can be set from 1-255. A boundary ("Ba"), which is used in "Fill", can be either any line or be of the same color as the "Fill" color. Zoom ("Zo") can be done by a numerical scale value or by position. Rotation ("Gr") can be done by degrees or by position. On those items in "Status" where there are two

possibilities, ("Zoom", for example), invoking the item toggles the result.

Colors are changed by invoking "Color" index, and then typing in the number of the desired color (0-15). The selected color appears in the left-most box (labelled Current Color) in whatever the current pattern index is (default is solid). The new color is transferred to "Line", "Zone", "Fill", "Marker" and "Background" by positioning the cursor in the appropriate box and hitting <DEL>. The "Pattern" index is also changed in this same way. The author selects "Pattern", then the index number (0-9), and moves the new pattern to the right area. The "Line" style is changed in much the same way by entering "Line" and then the style number (0-7). "Keep" saves the "Status" values. If the "Status" values are not kept, those changed values will only be maintained for a continuous session ("Quit" or "Discard" stops a session). There is only one "Status" board per file.

3. Creating an Object

As explained in Chapter II, the basic unit of a graphics frame is an object. Objects are created using the seven system primitives (line, circle, arc, etc.) or other objects. The author starts the creation process by invoking "Draw" on the second level command line.

The Draw command line is:

```
Lin Bez Mrk Zon Fl Cir Arc Obj X + - ? Grp Sty Rel Del
Ver Esc Quit
```

See [Refs. 1,2] for complete descriptions of all commands.

a. Example Object Creation

Figure 3.4 shows a simple arrangement of three boxes, a circle, and several dots. The order in which the objects are drawn on the screen is the same order which is used each time the frame is later displayed. (It is possible to modify the drawing order later, if the author determines that the original order is incorrect.) To draw these shapes, the author selects "Cir" (circle) and is prompted for the center point, and then for a point on the circle.

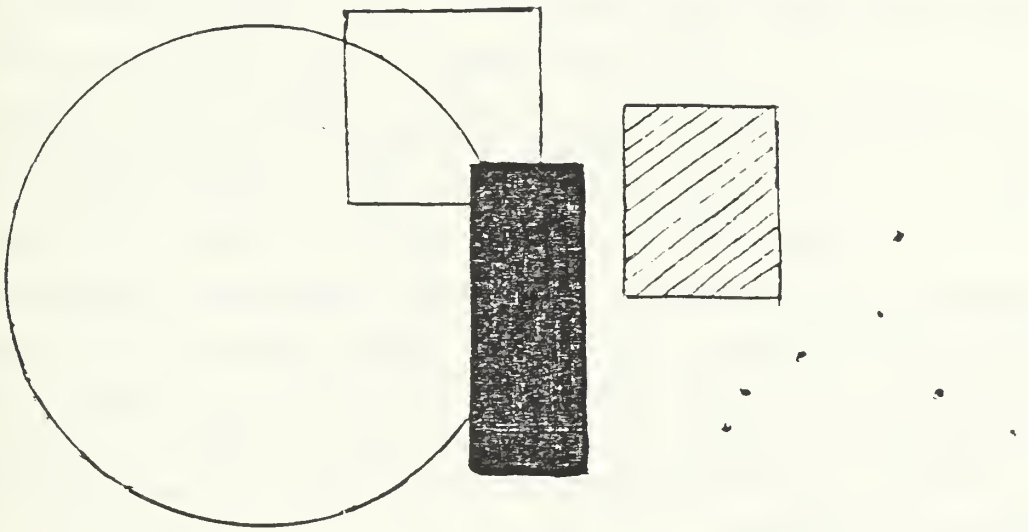


Figure 3.4 Sample Frame

These points are entered by positioning the cursor with the function keys and hitting <DEL>. The circle is drawn in the color specified in "Status" for lines.

The open box is formed by selecting "Lin" (Line). It can be drawn in two different ways. The picture on the screen is exactly the same, but one method results in four groups, and the other in one group. The former method positions the cursor on one vertex, enters the point using <DEL>, moves the cursor to the next vertex, again entering the point with <DEL>, and then re-selecting "Lin". This process is repeated three times (except for the last step of the third iteration). This results in four groups of one line (side) in each. The other method starts in one corner, enters the point with <DEL>, and then moves from corner to corner, entering the point each time with <DEL>. This results in one group with four lines in it. Whether the author chooses to use one method or another depends on the amount of later changes that may need to be made. The box made of four groups can be modified side by side, while the other box made of one group, can only be changed as a whole unit.

The third object to be drawn is done by selecting "Zon" (Zone). The author moves the cursor to one corner of the zone, enters it using <DEL>, moves to the diagonally opposite corner, and enters it with <DEL>, completing the zone. A large zone (about 1/4th of a screen) can take an appreciable amount of time to draw and probably should be avoided.

The third rectangle is a box created by drawing lines and then filling it with a pattern of diagonal lines. The box is drawn as previously described, then the author selects "Fl" (Fill). The cursor is moved to any point inside the box and entered using <DEL>. The box is then painted with the selected fill pattern and color. The fill proceeds until it encounters the boundary condition. This condition can either have the filling stop at a boundary of the same color as the fill, or at any boundary. The author

could inadvertently fill the whole screen if the wrong boundary condition had been set, or if there is a break in the boundary. "Fill" is slower than "Zone", and therefore should only be used in small areas (1/16th of screen or less).

The final group on the frame is a series of markers. The author selects "Mrk" (Marker) and then enters dots point by point with <DEL> (moving the cursor between points, of course).

b. Changing Styles

If the author needs to change the color of a line, perhaps for drawing the box that was filled in the example, this can be done on a temporary basis by using "Sty" (Style). The Style command line is:

```
Color Index Default Mode Pause Ver ? Quit Esc
```

When "Color" is selected, the color spectrum (as in Status) appears, in whatever pattern or style index is the Current Color as specified in "Status", temporarily replacing the command line. The author moves the cursor to the desired color and enters it using <DEL>. This does not affect the color kept in "Status" and the new color lasts only until a new option has been selected or until "Draw" has been exited, whichever comes first.

c. Additional Considerations

If the last action is a mistake, filling the whole screen, for example, "-" deletes the last object drawn. "+" will put back what "-" erased.

The principle for using "Arc" and "Bez" (Bezier curve) are similar to "Line", "Zone", etc. Bezier curves

require much practice in placing the control points properly to produce a recognizable and smooth curve.

At this point, the author may wish to make modifications in what is on the frame or to make it an object. This can only be initiated on the second level command line. "Quit" exits "Draw" and returns the author to this second level.

d. Naming the Object

At the second level, selecting "Create" provides the author with the opportunity to name the object. The "Create" command line is:

```
Frame Object Current-frame
```

"Object" asks if the current frame should be made into an object, and if yes, the author must provide a name for the object. Names should be unique, descriptive and comply with the naming conventions of the operating system (not more than eight characters long, etc.).

The author could have started the whole above process by naming the object, then drawing it, rather than as described. Selecting "Create", "Object" and then drawing the object, would have the same result as presented above.

4. Making a Frame

A frame is created either with named objects or by drawing the frame as a whole. "Create" frame is the default state for GRAFEDIT. The author is placed in that state upon entry to GRAFEDIT or after a frame has been saved.

The objects used on a frame can either be ones existing on that specific frame, or can be located on another frame, or in another file. This concept of a

storage or library frame is very useful as objects need not be constantly re-created. The author obtains objects from other frames by the "Get" ccmmand on the second level command line. "Get" asks for a filename, and a segment name if there is more than one segment in the file, then displays the command line:

```
Getfile Info Frame ? Quit
```

"Getfile" re-starts the process of getting a graphics file. "Frame" prompts the author for a number within the specified segment. After the frame is displayed on the screen, the command line is:

```
Object All_objects Merge Next Prev Getfile Info Frame
? Quit
```

"Object" shows the author a list of the available objects and the author selects one by typing first "Number", then the object's number. "All_objects" takes all the objects residing on the frame, while "Merge" takes the objects and the frame. Whatever has been selected, whether an object or a whole frame, is merged with the current frame. If some of the new objects have the same names as ones on the current frame, the author is advised of this, but as the objects are given unique numbers on the object list, nothing is lost or destroyed. The new objects are added to the end of the object list.

Once the needed objects have been obtained, the author makes the frame by invoking "Draw" and then "Object".

The "Object" first level command line is:

```
Use Copy ? Esc
```

"Use" displays the list of named objects with an asterisk beside those that have not yet been used, and the command line:

```
Choose an object: Number ? Esc
```

If there are more objects than can be displayed in one column, "More" appears on the command line, and selecting it, cycles the object list to the next column of numbers. The author selects "Number", then types the number of the desired object. When the "Draw" command line re-appears, the object name is written one line above it at the left side of the monitor screen. The object is placed on the frame by positioning the cursor and hitting <DEL>. The actual object position will have the same orientation to the cursor as it had to the center of the frame when created. The author can either draw all the objects in the same manner, and then modify them to create the complete frame, or each object can be modified before drawing another. The latter method is especially useful when objects have been created and stored as a full screen version, but which will be used in a reduced size version.

5. Viewing Other Frames

As mentioned in the section on TEXTEDIT, the author needs to make reference to the associated text frames to ensure proper object placement. This is accomplished at the second level command line with the command "Text". If no

text file has been specified in "Status", the prompt asks for the file name and the segment name (if more than one segment is in the named file). The "Text" command line is:

```
Getfile Segment Clear Frame Next Previous Quit
```

The explanation of these commands is the same as for viewing graphics files from TEXTEDIT.

The author may wish to see other graphics frames or objects. This is done by using the "View" command on the second level command line. The "View" command line is:

```
Local_object Names Frame Object Current_frame Esc
```

"Local_object" allows the author to view objects from the current frame, while "Object" allows the author to view objects on a different frame. "Frame" displays a frame from the current file or another file. "Names" displays object names and allows the author to change an object"s name. "Current_frame" re-displays the current frame.

As with TEXTEDIT, the author must first "Keep" a frame, in order to proceed to the next frame. If some objects were unused on the frame, the author is asked if they should be kept. The author exits GRAFEDIT by typing "Quit" on the second level command line. As in TEXTEDIT, the command line now is:

```
Save Discard Return
```

"Save" will keep the changes, and as with TEXTEDIT, it is acceptable to use the same name for the changed file as for the original one.

6. Changing Frames And Objects

a. Editing

Editing is initiated from the second level command line. The "Edit" command line is:

```
Frame Object Name Esc
```

"Edit Frame" will change an existing frame, either in the current file or in another file. Editing a frame means modifying the objects or groups that compose it. This first level of editing uses the highest level of objects on the frame. So if objects were used to create other objects before being drawn on the screen, only the outermost layer of objects will be available when "Edit Frame" is selected. The only changes are to the various objects as complete entities. That is, change in size or orientation of a whole object is possible, but not changing the color or size of any object that makes up the high level object. The author can edit an existing frame, then edit an object on that frame. "Edit Object" operates on objects in the current frame only. Once edited, all uses of that object on the current frame reflect the changes. "Edit" only changes the state away from "Create", the actual changes are accomplished through "Modify" or "Draw".

b. Modifying Commands

The Modify command line is:

```
Move Rot Ext Zoom Sty Fir Last  Bfore Del Win + - ?  X
Undo Add Ver Quit
```

When Modify is invoked, the current object or group is displayed to the left of the monitor screen on the line above the command line, and its control points are highlighted. The author can cycle through the objects or groups that compose the frame or object by using "+" or "-".

An object or group can be moved from one position tc another by using "Move". The first point required is the origin reference point. The second one gives the direction and distance of the move. The object is moved with respect to its position abcut the origin point.

Objects or groups can have their orientation shifted by using the "Rot" (Rotate) command. "Rotate" requires either three points cr two points and a degree amount. The first two points for either method set the origin and the reference line. The third point or the amount sets the degree of rotation.

Once the desired object or group has been selected, the author can change the size by using "Ext" (Extend) or "Zoom". "Extend" stretches the object cr group in the specified direction by the specified scale factor. "Extend"'s scale factor requires three points: the first is the anchor, the second sets the unit distance and the direction of the scaling, and the third gives the scale factor with respect to the anchor point. "Zoom" scales the whole object or group by the specified scale factor (a shift in two dimensions). After either of these two commands is selected, the author is prompted for the necessary values. The scale factor can be entered by position or by number. Both "Extend" and "Zoom" require an anchor point and a second point to determine the unit distance. The method for determining the scale factor is that specified in "Status" ("Zo" on "Status" command line). "Position" requires a third point to set the scale, while "Value" provides the actual scale factor. The methcd is previously selected in "Status" ("Gr" on the command line).

A group's attributes can be changed by using "Sty" (Style). The method is the same as was described for changing the style in "Draw".

"Del" (Delete) removes the current object or group. "Add" replaces what "Delete" erased. "Undo" erases the last change. The author returns to the command line by typing "Quit".

F. BUILDER

1. Structure

Lesson construction in BUILDER revolves around the concept of a path and its branch points. A path can be strictly linear with no branch points (similar to reading a book from cover to cover), or it can have branch points and offer many choices (like a "tree" structure). The complexity of a path increases in direct proportion to the number and type of branch points. A branch point is a place in the path where the student is required to make a choice. The choice can be in response to a question on the lesson material, or it can be a choice on what material the student wishes to see (choosing to use a Help segment, for example). Each path must end in one way or another, or an error or ambiguous situation results. Paths may end by jumping to a label ("GoTo"), "Join"ing the next step, "Halt"ing (ends the lesson session), "Exit"ing (ends the lesson segment), or "Repeat"ing the decision point (Figure 3.5).

This section will explain how to build two different structures: a menu or control segment (Figure 3.6) and a short called segment (Figure 3.7).

2. Initial Steps

BUILDER is invoked by typing BUILDER<RET>. The author must ensure that sufficient space is available to

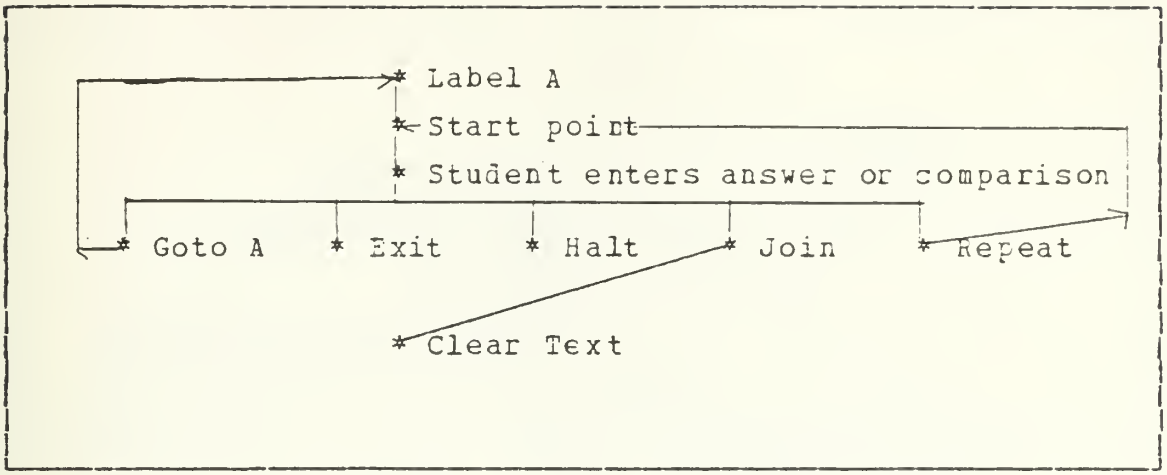


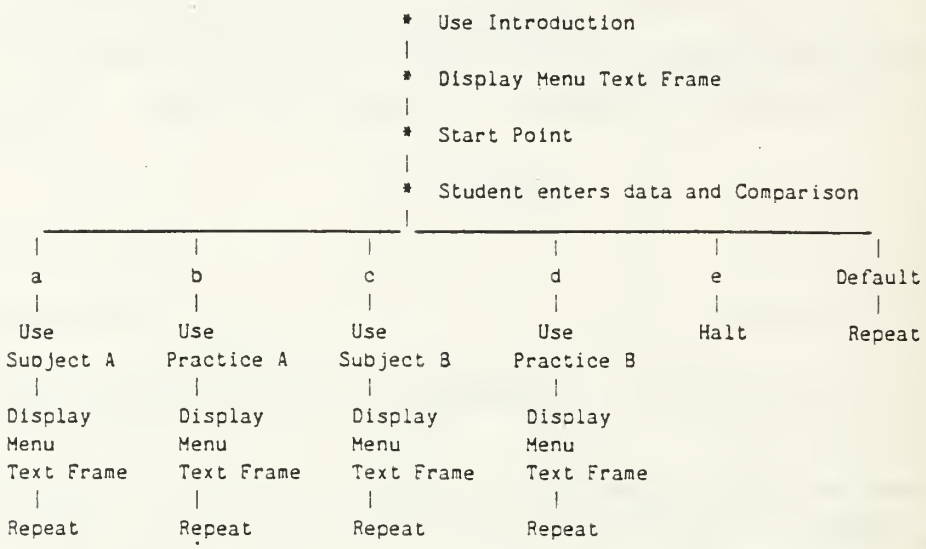
Figure 3.5 Examples of Path Terminators

store the four files that BUILDER creates (.RUN, .TXT, .PAG, and .GRF). BUILDER itself requires 143K bytes of space. For example, a short four frame lesson with no graphics required 8K bytes of space, and a menu with eight called segments needed 16K bytes.

A text and a graphics file are needed to create a BUILDER lesson segment. Even if the graphics file is not needed, BUILDER allocates 8K bytes despite having no graphics in it. It is, therefore, more space efficient to have a graphics file, even if it is not used in the lesson. The text and graphics files required for BUILDER will probably not fit on the same diskette as the BUILDER program and the four lesson files.

A BUILDER convention requires that the lesson name and the segment name be the same. The lesson name is what will be the file name with the four extensions mentioned above. The segment name is what is referenced by calls within the lesson.

Once BUILDER has been invoked, the first prompt is for the lesson name. If a lesson of that name already exists, the author is asked, in turn, if each of the four



[Ref. 2: p. 15],

Figure 3.6 Menu Segment

lesson files should be replaced. A negative response will result in a request for a new lesson name. The second prompt is for the segment name, which should be the same as the lesson name. The third and fourth requests are for the text and graphics files, respectively. If either has more than one segment, the segment name is requested.

The first level BUILDER command line is:

```
Branch Clear Display Remember Use <.>
File Status Wait View  Prompt Quit <.>
```

```
        *                   Display-Text frame
        |
        *                   Prompt student by flashing
        |
        *                   Display-Graphics frame
        |
        *                   Wait for 10 seconds
        |
        *                   Clear-Portion
        |
        *                   Wait for student to type any key
```

[Ref. 2: p. 2],

Figure 3.7 Called Segment

The period, <.>, toggles the command line. Most commands can be aborted, once invoked, by <ESC>.

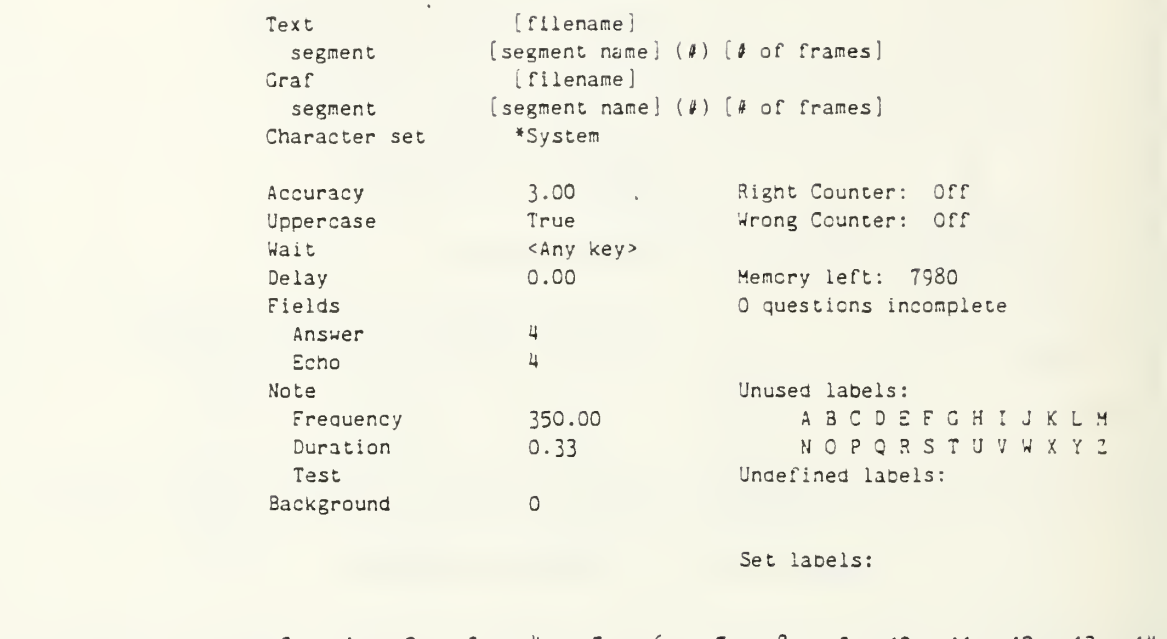
3. Building a Lesson

a. Set Up

Before the lesson building actually begins, the author may need to alter the default values in BUILDER. "Status" displays the default values (Figure 3.8). The "Status" command line is:

```
Text Graf Char Accur Upper  Wait Field Delay Note Back
Exit
```

As mentioned in TEXTEDIT, BUILDER contains the background color specifications. The author can change the color by selecting "Back" (Background), and typing the number (0-7) of the desired color. The default color is black.



[Ref. 2: p. 53],

Figure 3.8 BUILDER Status Board

"Accur" (Accuracy) is measured in percent wrong and ranges from 0.02 to 100. It refers to the student's response or choice at a branch point or after a question. "Note" has the frequency in Hz and time of the prompt duration in seconds. The "Freq"uency range is 0-20 KHz, and the note "Dur"ation is 0.034-60.0 seconds. "Exit" returns the author to the first level command line.

During the lesson, the author can, and should, enter "Status" to check which labels have been properly set, and which have been used without being defined. Undefined

labels result in undetermined paths, or in other words, an error.

b. Displaying and Clearing Text and Graphics Frames

Generally it is a good idea to both begin and end a segment with a clear screen. This avoids inadvertent overlaying of frames. "Clear" has the command line:

```
Text Graphics Both Answer Reply Counter
```

The "Clear Text" and "Clear Graphics" commands are used when only the text or graphics portion of a screen need to be erased. Text clearing is done line-by-line from top to bottom, and so is rather slow. "Clear Both" is faster than "Clear Text" and can be used even if there are no graphics present on the screen.

Text and graphics are put on the screen using the "Display" command. The "Display" command line is:

```
Text Graphics Echo Counter Reply Video
```

Both the "Text" and "Graphics" commands, when invoked, display the range of frames present in the specified segment and request a frame number. When the requested frame is displayed, the author must confirm that the frame shown is the one desired. If it is not the right frame, another frame can be requested at that time without leaving "Display Text" or "Display Graphics".

Text and graphics frames may be overlaid to create the lesson frame. The order in which they are displayed is totally dependent on the text and graphics frame composition. Overall, it is probably better to show

text first, as it is displayed more rapidly than graphics, and the student has something to look at almost immediately. Use of text labelling on graphics frames, however, usually requires that the graphics be displayed first (labelling of inputs, outputs, ground and power in a circuit diagram, for example). Often labels are overlaid directly on figures, and due to the opacity of VCIS frames, the sequence must be graphics, then text in order for the text to be visible.

Once displayed, the text and graphics should remain available for the student to read for a certain period of time. This period of time may be set by use of "Wait". The "Wait" command line is:

```
Default Any_key Frame Key System Time
```

"Any_key", "Key", and possibly "Default" (depends on what "Default" has been set on in "Status"), require the author to specify a cursor display position. The bottom right corner of the monitor screen displays the line and column number of the cursor position. "Key" has the author select any one key that the student must depress before the lesson continues. "Time" sets a period of time from 0.1 to 300.0 seconds. This is most useful in displaying the logo or title frames. "System" is similar to "Any_key" and "Key". It uses <SPACE BAR> and does not require specification of a cursor display position.

c. Using a Lesson Segment

One way to minimize building complexity, and to ease editing or modifying of lessons, is to split long lessons into segments. This is also useful in creating Help segments and in menu structures.

The "Use" command line is:

```
Graphics Text Special Lesson_Segment
```

"Graphics" and "Text" store frames for use in a "special" lesson segment. "Special" calls a Pascal program written by the author to supplement VCIS. "Lesson_segment" prompts the author for the segment name. This can be used to call a Help segment or as part of a menu structure, where when the student selects a certain option, the result is a call (use) of a lesson segment (see Figure 3.6).

d. Branch Points

The actual structure of a lesson is determined by use of various branching techniques. Only selected branching instructions will be explained in depth here. See [Ref. 2] for further details. The "Branch" command line is:

```
Keyboard Counter Memory - Label  - GoTo Exit Halt Join
Repeat
```

"Keyboard", "Counter", and "Memory" define where the decision information on a branch is to be obtained. The other six commands are either path terminators, or are used in path termination, and are explained later.

The "Keyboard" command line is:

```
Answer Character Position - Timeout
```

"Answer" first has the author select the area of the monitor screen where the student enters his/her response. The left

edge of the space is selected first, (using the function keys to move the cursor and <DEL> to select it), and then the right edge. This space may be at most one line long. The column and line position of the cursor is displayed in the bottom right corner of the screen. The type of "Answer" that may be put in that space is selected from the command line:

```
Neutral Right Wrong Unmarked - Switch
```

Selection of "Neutral", "Right", and "Wrong" requires construction of a path. The next command line is:

```
String Pattern Numeric Missing Unanticipated
```

The author must build the expected answers, whether right, wrong, or neutral, first, before doing the unanticipated answer. The unanticipated response is used as the default and must be constructed last.

A "String" answer is composed of up to 75 characters, which may include the logical operators AND (&), OR ( ) and NOT)(. Sample strings can be tested using SCOMPARE (before or after BUILDER session) to verify a match.

Choosing "Keyboard Position" is useful in creating a menu structure. The author must first mark the cursor display position using <DEL>. As with "Answer", the next command line is:

```
Neutral Right Wrong Unmarked - Switch
```

"Neutral", "Right", and "Wrong" all lead to:

```
Anticipated Missing Unanticipated
```

"Anticipated" has the author mark (with <DEL>) first one corner and then the diagonally opposite corner.

Once the corners are marked or the string is specified, the author continues the path that that particular student response has chosen. At this point, the path may use a segment, display text or graphics frames, etc. The path ends with another invocation of "Branch".

e. Path Terminators

Path terminators are selected from the "Branch" command line:

```
Keyboard Counter Memory - Label  - Goto Exit Halt Join
Repeat
```

"Halt" is only used to terminate the lesson session, while "Exit" is used to end a lesson segment. "Goto" requires a label to which to branch, and "Label" sets this point. If an author wants the student to see a certain frame, the label should be placed before the frame is displayed. The student will not see the frame, if the label is defined after the frame has been displayed, even though the clear command has not been issued. "Join" simply continues on to the next command after the branch point. "Repeat" returns the student to the last decision point. It can be used for an unanticipated answer or for the path default. The author is asked to confirm the path terminator in all cases before the next command is accepted.

f. Ending a Segment

As mentioned in path terminators, the ways to end a segment are with a "Goto", "Join", "Exit", "Halt", or "Repeat". As the path progresses, there is a number or "Main" to the left of the command line. This indicates which path the author is creating. For example "2.3.7" would indicate that the author is on the second path from the main branch, the third path down the second route, and on the seventh branch from the third. Any time a default path is being built, the number terminates in a "D" ("1.D", for example).

4. Editing a Segment

There is no editing capability in the BUILDER program similar to that of TEXTEDIT or GRAFEDIT. Changing or modifying a segment is either done by brute force (totally redoing the entire segment) or through use of files. The BUILDER command file (.TXT) can be edited using a text editor and then used instead of keyboard input. The command "File" gives the command line:

```
Breakpoint Comment Input
```

"Input" calls this command file. If changes are necessary in the graphics or text files, but not in the command file, the author has to rebuild the segment but can use "File" to produce the command file.

G. INTERP - TEST MODE

1. Initial Steps

When used in the test mode, INTERP is invoked by INTERP<RET>. There are no files created nor is any information recorded in test mode. INTERP is 157K bytes and so it may not be possible to put the segment to be tested on the same diskette. When the lesson name to be tested is requested by INTERP, if other than the default (same drive as INTERP), the actual location of the test segment must be specified. INTERP must have the four files generated by BUILDER (.RUN, .PAG, .GRF, and .TXT) for each segment or the three files created by LINKER in joining segments together (.RUN, .GRF and .PAG).

Upon invocation, INTERP asks for the lesson name, and if tracing information is to be displayed. The tracing information shows the question and path number, and text and graphics frame number (not as created in TEXTEDIT or GRAFEDIT, but numbered sequentially from the beginning of the segment).

2. Segment Testing

There are only two major differences between the testing and the final mode of lesson operation. One is that in the testing mode, the lesson management conditions supplied by MANAGER.DAT are not available. The other is when a segment is called and is not linked to the segment actually under test. INTERP returns a message stating "Calling segment" and continues the segment being tested. It is therefore possible to test all segments created by BUILDER, whether the segment is a menu or control segment which calls other segments, or the segment is a called segment. A correctly built segment operated under INTERP should operate exactly as if it were being actually used by a student.

The segment being tested can be terminated either as specified by the author or in a less graceful manner by <ESC>. The escape gives the command line:

```
Resume Stop Backup Comment
```

This escape mechanism is always available, even during the final mode, but it is intended as an emergency exit and as a means of allowing student comment or backup. The <ESC> works when INTERP is waiting for a response. It will not work, for example, if INTERP is in the middle of a timed wait period or in the midst of a text or graphics display process.

H. LINKER

1. Initial Steps

LINKER is invoked by LINKER<RET>. The author can specify only one control segment and eight called segments. It is possible to have a called segment itself be a control segment which calls other segments. Another possibility is using one version of the lesson as the control segment, with the called segments being corrected versions of segments in the "control" portion.

2. Linking Segments

When invoked, the first prompt in LINKER is for the host segment (or the control segment). The following eight prompts are for called segments. Any or all of the segments may be on a diskette in a drive other than the one occupied by the LINKER diskette. If less than eight called segments are used, the author types <RET> to continue. The next prompt is for the name of the output segment, which may also

be placed on a diskette different from that of LINKER. If there is already an output segment of the same name, the author is asked if it should be replaced. Segment names only must be used - if a lesson name is used, the linking procedure will not work. LINKER displays the names of the segments as it is linking them. If one segment calls others, they also are listed.

3. Possible Errors

If a segment cannot be found (for example, because of a misspelled name), LINKER returns and states that the segment cannot be found. If the file (lesson) name and the segment name are not the same, there may not be an error message, but when LINKER displays the names of the linked segments, the incorrectly named segment will not appear, as it has not been linked. If something occurs to cause LINKER to terminate, and if there is no apparent reason for the termination, LINKER should be executed again. The author may need to enter the called segments in a different order. The order entered in LINKER does not affect the final version.

I. MANAGER

MANAGER is invoked by typing MANAGER<RET>. It does not require the presence of any files. When MANAGER is invoked, the first prompt is "Using which directory?" It is best to use the same directory or disk drive upon which MANAGER is residing. If MANAGER.DAT is not present, it is created. If no reply file or if no student roster is kept, the initial MANAGER.DAT file is 6K bytes.

If there is a MANAGER.DAT file present in the directory, the command line is:

```
Status Names Replies List Quit
```

If MANAGER.DAT was just created, the system is placed immediately into "Status". The default values for the lesson management conditions in MANAGER are shown in Figure 3.9.

```
STATUS:  <A..L> <RET> <ESC>
A.  videodisk/tape ---- F
B.  halt ------------- F
C.  record path ------- F
D.  unanticipated ----- F
E.  backup ------------ T
F.  use roster -------- F
G.  use passwords ----- F
H.  use student ID # -- F
I.  use replies ------- F
J.  segment and frame
K.  string locations:
L.  lesson name:
```

Figure 3.9 Manager Status Board

They are changed by typing the letter of the line and then typing either "T" or "F". A "True" response means that lesson management condition is allowed. The lesson name must be entered and it can indicate that the lesson resides on a disk or drive that is different from MANAGER.DAT. If passwords, replies, roster or ID numbers are to be used, the author must supply them. The changes in "Status" are kept and the author is returned to the command line with <RET>. The roster, ID numbers, and passwords are entered after invoking "Names". Replies are entered after invoking "Replies". "List" changes the student roster. When "Quit"

is invoked, the author is first asked if the file (MANAGER.DAT) should be updated, and then if another directory is to be done. The response each time is "Y"<RET> or "N"<RET>.

J. INTERP - FINAL MODE

This is the final version of the tutorial as seen by the student. INTERP, MANAGER.DAT, and the lesson file specified in MANAGER.DAT must be present, although not necessarily on the same diskette. If any of the lesson management conditions have been specified and if the lesson itself is large, there probably will not be enough room on one diskette.

The lesson can be invoked first by INTERP<RET>. INTERP should be changed to something a little more descriptive of the tutorial before it is presented to the student, however. Under the Microsoft Dos, this is done by "REN oldfilename.extension newfilename.extension".

K. LISTFRAM

LISTFRAM is the program used to display text or graphics frames at the terminal. Text frames can be converted from a non-ASCII file to an ASCII file which is suitable for printing.

The program is invoked by LISTFRAM<RET>. The command line is:

```
Text Graphics Quit
```

When either "Text" or "Graphics" is selected, the procedure is the same. The next prompt is for the filename. The file may reside on a drive or diskette that is different from LISTFRAM. As LISTFRAM is 70K bytes, it is possible to have

the file to be listed on the same diskette as the listing program.

The next command line is:

```
All Frame Getfile Help List Output Part Quit Segment
```

If there is more than one segment in the file, "Segment" must be invoked. "Part" lists part of the file. The first prompt is for the starting frame number and the second prompt is for the ending frame number. "All" starts with the first frame and lists all of the frames in the segment. In either of these commands, the command line for each displayed frame is:

```
Next Previous Quit
```

The clearing of the screen between frames is taken care of by LISTFRAM.

"Getfile" is used to obtain another text or graphics file for listing. It basically re-initializes LISTFRAM in text or graphics listing mode, whichever was first selected upon invocation of LISTFRAME. To switch from listing graphics to text or vice versa, the author must return to the first level LISTFRAM command line. "List" displays information on the file previcusly specified in the same format used in either TEXTEDIT or GRAFEDIT. "Help", as in previously described VCIS programs, displays a frame containing one line descriptions of LISTFRAM commands. "Frame" asks for one frame number and then displays that frame.

At this time, only text frames can be printed out. The University of Utah programmers are working on a program for

the IBM PC that will print graphics files. Text files are printed by invoking "Cutput". The new command line is:

```
Console Disk
```

The author picks "Disk" and is then prompted for an output filename. This name may be the same as the TEXTEDIT file as the extensions are different. Selecting "Disk" changes the default listing location for future LISTFRAM commands. Now, by invoking "All", the text file is converted and listed into the disk. The actual printing is done after LISTFRAM has been exited with "Quit". The print command for the IBM PC is "prn filename.extension".

L. SUMMARY

This chapter has been included to provide the prospective author with a basic knowledge of VCIS commands. It is not intended to replace the User's Guides, and should be read in conjunction with the Guides. The following chapter shows how this authoring system was used to create two tutorials.

IV. CREATION OF VLSI TUTORIALS

A. INTRODUCTION

All of the VLSI computer-aided design (CAD) tools at the Naval Postgraduate School reside at present on the VAX 11-780 operating under the Berkeley UNIX 4.2 operating system. As there were no short easy tutorials available for first time users to become acquainted with UNIX and one of its text editors, vi, it was decided that UNIX would be the first tutorial. This tutorial does not require the use of graphics and so eases the learning process involved with a complicated authoring system like VCIS. The second tutorial then introduces the student to the world of VLSI - through its shapes, colors and notations. Graphics is a major portion of this tutorial. The two tutorials together exercise most of the functions or options of VCIS. No video segments were used, as it was decided that the material was not suitable for video. A copy of the UNIX tutorial is provided in Appendix A and a copy of selected portions of the VLSI tutorial is in Appendix B.

B. LESSON PLANNING

1. Structure

One of the first decisions made was to use a menu-driven structure. Students do not always have the solid block of time needed to complete one long sequence of text and questions, nor is it easy for them to start in the middle of a long sequence after a lapse of time. The menu allows them to choose to do those sections for which they have time, or which they wish to review before continuing.

A menu-driven structure also lends itself very well to division into segments which can be more easily built and modified.

The VLSI tutorial has not only the main menu, but two embedded menus. The author decided that the amount and type of material would fit together better in this manner. The section on inverters, NAND, and NOR gates is a lengthy section if done as a whole, but is more manageable for both the author and the student when separated into three subsections.

2. Questions

The next decisions made were on the type of questions. The first tutorial on UNIX needed questions that required the student to actually perform the commands rather than memorize a response. This was achieved by using a two screen approach where the student logged onto the UNIX operating system at the same time he/she "booted up" the lesson on the IBM PC. The student was provided with a selection of files for practicing file manipulations, as well as text editing.

The second tutorial did not use questions at all as it was decided since the questions that could be asked would be trivial, it would be better not to ask any. It was assumed also that a student doing this tutorial would also be taking a class in VLSI design and so would be doing problems/questions for the class. The format for the shapes, colors and notations tutorial became that of screens of text. The student is presented with a design problem upon completion of the tutorial that is intended to be a paper-and-pencil design checked by the instructor.

3. Assistance

The first tutorial required that HELP be available for each command under instruction. There is no on-line help available in vi (the text editor) and while there is assistance in the main level of the UNIX operating system, it would be disruptive to the lesson to have the student accessing it. As the student was not answering questions in the second tutorial, help does not need to be available.

C. TEXT

1. Text Composition

For both tutorials, the text was edited to ensure that each frame was complete in itself with little need to refer to a previous frame. The author also ensured that the amount of text on the screen was not excessive. This is of particular importance in the VLSI shapes, colors and notations tutorial where graphics cccupy a significant portion of the screen.

The UNIX tutorials are designed to present the minimum amount of knowledge a student needs to get on the system and do limited file manipulation. The student can either do selected sections for a bare minimum of knowledge or do the entire tutorial to gain knowledge of a few more complicated commands that do the same processes in another manner. It does not attempt to cover all commands or all options within a specific command, but is only meant to provide a means for the student to start on the UNIX.

The VLSI tutorial follows [Ref. 3]. This was used because the author assumed that the text used for a VLSI class would most likely be this one, and so would not present any conflict with material presented in class.

2. File Order

The principles and actions for entering or changing text were the same for both tutorials. A copy of the first tutorial is listed in Appendix A. Wherever possible, the author attempted to maintain the tutorial text in sequential order according to segments. That is, a control section (the menu) is done first, followed by sections called by the menu. Each section is preceded by a chapter title page.

3. Attribute Selection

The process of attribute selection was basically the same for both the UNIX and the VLSI tutorials. The only difference is that the attributes selected were different, as the author wanted to see the effect of different colors in the presentation of the material.

When TEXTEDIT was first entered, the author edited the working set of colors to achieve a harmonious grouping of colors. Yellow, white and red were used as highlighting colors, with blue, green and gray used as primary text colors. Gray was used in the VLSI tutorial to contrast with the colors used to identify the various layers (for example, red is used for polysilicon). There was one set of colors used on the menu for cursor positioning instructions, and other colors for section names and selection instructions. Each frame had a section title at the top that was composed of one of the text attributes having a different background color from the rest of the frame (black was always used as the lesson background color). The frame title for help and summary frames was done with a different attribute than the rest of the section to emphasis the nature of the frame. At least one color having an underlining was selected.

4. Text Input

The process for text input for the two tutorials was exactly the same. The only significant difference was due to the use of graphics in the VLSI tutorial. This necessitated constant checking between the text and graphics to ensure that nothing had been covered up and that the frame was not crowded.

A text frame was also created by overlapping previously created frames. This was of particular use in the menu segments, where the only difference between one frame and another, was the section of the menu which had been highlighted.

5. Text Modifications

As before the only difference between modifying text in the UNIX tutorial, and in the VLSI tutorial, was the graphics used in the latter one. Large scale modification of text was not possible without constant reference to the graphics frame.

D. GRAPHICS

1. Object Selection

The VLSI tutorial is the only one of the two that uses graphics as a vital ingredient. It was not appropriate to use any graphics to describe UNIX. The objects selected for the VLSI tutorial were taken from [Ref. 3]. They were also chosen for their simplicity - both for ease in creation and ease in comprehension.

2. Object and Frame Creation

The basic unit of a graphics frame is the object. For example, the layout notation for an inverter was done by

successively choosing a color and a pattern (overlapping solid colors do not show underlying colors) and "Draw"ing a "Zone" of the desired size. The color and pattern selection were initially done in "Status", thereafter, requesting a different "Style", temporarily altered the "Color" and or the "Pattern". At certain times, the author stopped to designate what had been "Create"d as an "Object". An object was named as such by the author when it could be used to create another object. For example, an inverter (an object) with another input added became a NAND gate (another object). Selected objects were created once and then moved from one frame to the next when needed.

After each individual object for a given frame had either been created or obtained from a previous frame, the whole frame was constructed by "Draw"ing "Object"s one at a time where needed. If the size was wrong, it was "Modifi"ed and "Zoom"ed larger or smalled as needed. It was also possible to "Ext"end (larger or smaller) an object in either the vertical or horizontal dimensions. When the frame was complete, the author only saved objects actually used in that frame. One frame was kept as a library frame which contained most of the basic objects.

3. Object and Frame Modification

Either the frame or the object were modified, depending on the degree of change required. If an object was modified (size or color changed, for example), it was changed wherever it appeared on the frame.

Object and frame modification occupied a large portion of the author's time on the VLSI tutorial, as each time text changed, a corresponding change in the graphics was often required.

E. BUILDING THE LESSON

1. Using BUILDER

The sequences used to build the two tutorials were essentially the same. In each the menu segments were built and tested first, followed by the called segments. As mentioned previously, only the VLSI tutorial has graphics. The VLSI tutorial also has embedded menus but those were created in the same manner as a main menu and linked in the same way.

The lessons were usually created by "Display"ing first "Text" and then "Graphics" (except no graphics in UNIX tutorial). In those instances in the VLSI, tutorial where the graphics would have covered up the labels, the above order was reversed. The lesson either "Wait"ed for a certain period of "Time" or for a signal from the student (usually <RET>) before progressing on to the next stage of the lesson. Based on student surveys, the author determined that a signal from the student was preferred, since reading rates varied as did the possibility of interruption. The "Clear Both" command produced a rapid clearing of the screen and so it was used even when no graphics were present. ("Clear Text" does so line by line while "Clear Both" clears the whole screen simultaneously.)

When a frame asked a question or required a choice, a "Branch" point was needed. For example, when the student selects a section on the menu, it is done by "Position"ing the cursor. An invalid or "Unanticipated" response presents the same screen again and again until a correct response (in this case, a correct position) is obtained.

Once a correct section was made, the author "Use"d another "Lesson_segment" (called a separately created segment). At the completion of the called segment, the student was returned to the main or section menu, with the

most recently completed section highlighted. The author briefly considered structuring the menu to postion the cursor on the next section on the menu after a given section was completed. This was not done, as the number of possible paths rapidly becomes unwieldy.

In the UNIX tutorial, the student could choose to go ahead to the next frame, go tc a help frame and rarely to back-up to a previous frame. This choice was also provided through "Branch"ing by a certain "Answer" provided on the "Keyboard". A "Right" answer got the indicated response, while a "Wrong" or "Unanticipated" answer repeated the frame. Each possible path at a "Branch" point was terminated properly.

In the UNIX tutorial, when the student "Halt"ed the tutorial, the possibility for comment was provided and the PC would "Remember" the student comment. This appeared to cause some confusion with the students and so was not used in the VLSI tutorial.

Each tutorial was tested by segment, menu-linked segments, and in total. The menu control segments were tested before they were linked to the called segments. They showed correct operation if the statement "Calling segment" was displayed when the USE statement was encountered. Each called segment was tested to show that it branched correctly at the desired points and responded properly to wrong answers.

The menu segments were connected together eight at a time using LINKER. The author had to ensure that sufficient disk space was available to hold the completed lesson. The completed VLSI tutorial is too large (130K) to fit on the same disk as all its component parts. When the segments were all linked, the completed lesson was again tested using INTERP.

2. Final Lesson Made with MANAGER and INTERP

After each tutorial had been tested, its MANAGER.DAT file was created using MANAGER. In the UNIX tutorial, the only special lesson management condition requested was recording student path and response. The VLSI tutorial used none of the special lesson management conditions provided as problems occurred that appeared to be linked to these functions.

When complete, the command name, INTERP, was changed to something more descriptive of the individual lesson - VIUNIX for the UNIX tutorial and SHAPES for the VLSI shapes, colors and notation tutorial. The student then uses the name VIUNIX or SHAPES to do the tutorial.

F. STUDENT COMMENTS

The author used a student opinion form on the first tutorial (see Appendix C) to help in the structuring of the second tutorial. Student comments on the UNIX tutorial were generally favorable. The author had asked questions on the effect of colors in text and if waiting for a signal was preferred over automatic change of frames. The students had no comments or criticisms on color selection. They did, however, prefer the use of a signal to an automatic change of frames.

V. EVALUATION OF VCIS

A. INTRODUCTION

VCIS provides a flexible and comprehensive means of producing tutorials. The current versions of the programs are very user friendly. It is possible to learn the system by using only the documentation and the help provided with each program.

B. GRAFEDIT

GRAFEDIT is a program that makes the creation and modification of graphics frames easy and relatively painless. A feature that is especially nice, is the ability to change a color or pattern within the DRAWing context without suffering the disruption of going into the STATUS board to make the changes. Another welcome feature is the ability to cycle within the objects on a frame or within groups within an object during the modification process.

C. VCIS AVAILABILITY

VCIS documentation lists several micro- and minicomputers that can use VCIS. The author can speak from experience on only two of these - the IBM PC (with Tecmar Graphics Master Color Board) and the VAX 11-780 with Berkeley UNIX 4.2 operating system. The IBM PC version is vastly superior to the VAX version that was available in the spring of 1984.

Part of the problem is due to the time sharing environment of the 11-780, of course, and cannot and should not be attributed to VCIS. The rest is due to emphasis by the

University of Utah on the microcomputer versions of VCIS. This is an entirely valid perspective for this authoring system as the microcomputer environment is much more responsive to the needs of the author.

D. COLOR SPECTRUM CONFLICTS

At present, the most annoying problem left in VCIS, and it is admittedly a minor one, from the author's point of view, is the discrepancy between the colors when viewing a frame produced by one program in the context of another. The color spectrum has eight darker, more vivid colors on one side and eight lighter colors on the other side (16 colors on an IBM PC with Tecmar Graphics Master Color Board). The order (light vs. dark) varies from program to program. There are two examples of this. The first example is the creation of a graphics frame using a dark vivid red and green. When this frame is viewed by either TEXTEDIT or BUILDER it appears as a pale red and green, and when seen in the final product (INTERP), is back to the colors in which the frame was created. The second example is in TEXTEDIT, where the author must take care to select, for example, a light blue foreground without underlining and a dark blue foreground with underlining, in order to present a uniformly light blue foreground in the final product. Neither of these problems is monumental, both are correctable, but together they represent a constant irritant to the author, who is never quite sure what the graphics or text frame <u>really</u> looks like. The University of Utah Computer Science Department is aware of this problem and it will be corrected in subsequent versions of VCIS.

Another color problem, which is a matter of taste rather than actual error, is the color selection in TEXTEDIT. With 16 possible colors (on the IBM PC using the TECMAR Graphics

Master Color Board), there are a number of color combinations provided that are probably unusable. Many of the lighter colors (light blue and light green, for example) are not visible against other lighter colors. In addition, the author can visualize no usage whatsoever for a foreground and a background of the same color (the letters are not visible). Again, the University of Utah programmers are aware of this and the color combinations will be rearranged to avoid the less usable combinations.

E. LINKER

The current version of LINKER generally performs as required. It links one control segment and up to eight called segments. However, the author encountered one totally unexplainable situation when the order of the called segments came into question. The order should not matter as LINKER should simply continue for the author-supplied list of segments until the next segment required by the control has been encountered. The author's actual and correct sequence of segments was, after the main control segment, an embedded menu with called segments, a normal segment, an embedded menu with called segments, and two more normal segments. LINKER continued to fail with errors indicating a problem within the normal segment located between the two embedded menus. The author relinked the menu segments, rebuilt the normal segment and reran LINKER without success. A University of Utah programmer suggested changing the order of the segments as given to LINKER which would not affect the final lesson order. The new order, which worked successfully, was the two embedded menus, followed by the three normal segments. While changing the order was an easy solution to the problem, it was not necessarily obvious from the error messages received.

F. NAMING CONVENTIONS

BUILDER requires the use of both a segment name and a lesson name. It would seem reasonable to have one lesson name and possibly several segment names. A segment name and a lesson name must be the same, as the only name under which the file is stored is the outer or lesson name. Reuse of a lesson name, even with a different segment name results in destruction or change to the previous segment. LINKER also uses segment names rather than lesson names during the linking process. The documentation on VCIS does tell the author to use the same name for both segment and lesson, but it creates minor author confusion to have two different labels for the same name.

G. MANAGER

MANAGER provides many potentially useful lesson management conditions. It provides the ability to monitor student trouble spots by recording their paths through and responses to the lesson. Unfortunately, the program to read the student responses and paths is not yet available. If students make comments or continue to experience difficulties in the same areas, they must still contact the instructor as MANAGER, at present, has write-only capability.

When the path or response recording function is used, the diskette obviously cannot be write-protected. While many students are responsible and probably would not deliberately seek to destroy a lesson, the possiblity for inadvertent destruction exists. The student has the frustration of not knowing exactly what has occurred and must obtain new lesson diskettes from the instructor. Exactly this situation occurred more than once with the UNIX tutorial. The author, therefore decided, that while recording student

paths and responses was a very useful lesson management concept, the concept was not yet ready for actual usage.

H. OVERALL COMMENTS

The author found the VCIS authoring system to be a very effective means of producing a high quality product. The support rendered by the University of Utah staff was magnificent, with questions answered and solutions to problems provided almost instantaneously.

#1 -- ONE -- B:VIUNIX

UNIX AND VI TUTORIAL

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

3 SEPTEMBER 1984

#2 -- ONE -- B:VIUNIX

This tutorial is separated into 8 sections which are accessed through a menu. Sections A, B, C, D, and H will provide the minimum level of knowledge necessary to create, edit, remove, print files and to get help. These sections should take about 45 minutes to complete. The remaining sections provide other and usually faster ways to do the same things. Those sections should take an additional 30 minutes.

The commands presented in this tutorial are only a small subset of those available under UNIX. Also each command has more options and uses than are presented here. The UNIX manuals are listed in section H for further reference.

#3 -- ONE -- B:VIUNIX

MENU

POSITION CURSOR UNDER DESIRED SECTION
LETTER USING ARROW KEYS AND <RET>.

A. TUTORIAL CONVENTIONS

B. UNIX CONVENTIONS

C. CREATING, EDITING, AND SAVING FILES

D. FILE MANIPULATION

E. MORE ON CURSOR MOVEMENT

F. MORE ON EDITING FILES

G. MORE ON SAVING FILES

H. OTHER TUTORIALS AND ON-LINE HELP

I. EXIT TUTORIAL

ARROW KEYS ARE FOUND IN THE
LEFT BOTTOM CORNER OF THE
KEYBOARD.
  F7 IS UP.
  F8 IS DOWN.
  F9 IS LEFT.
  F10 IS RIGHT.

TUTORIAL CONVENTIONS

#5 -- ONE -- B:VIUNIX

TUTORIAL CONVENTIONS

1. This is a dual screen presentation using an IBM PC for the tutorial and any terminal having access to the UNIX operating system for student responses.

2. Commands are entered exactly as shown.

3. Angle brackets < > are used as delimiters of input and are not to be typed.

4. <lowercase input> is the sample format for input following a command,

   e.g. vi <filename>.

5. <UPPERCASE INPUT> indicates one keystroke,

   e.g. <RET>.

Type <RET> when ready to continue with the tutorial.

#6 -- ONE -- B:VIUNIX

TUTORIAL CONVENTIONS

6. The control key, found on the left side of the keyboard on most terminals, is indicated by " ^ ",

   e.g. ^F.

   This is also considered one keystroke.

7. HELP can be obtained on each frame by following the instructions given.

8. A typing or spelling error can be erased by <BACKSPACE> provided that <RET> has not already been entered.

9. Each section concludes with a summary of the commands learned in that particular section.

Type <RET> when ready
to return to the menu.

#7 -- ONE -- B:VIUNIX

UNIX CONVENTIONS

#8 -- ONE -- B:VIUNIX

UNIX CONVENTIONS

1. UNIX uses both upper- and lowercase letters in commands and filenames. test.1, Test.1, and TEST.1 are three different filenames.

2. A filename may contain up tp 8 alphanumeric characters and must start with a letter. It may also contain up to 8 alphanumeric characters. The filetype is separated from the filename by the delimiter " . ". The filename and filetype together must not total more than 14 characters.

   e.g. page.txt PadsIn

Type <RET> when ready
to return to the menu.

#9 -- ONE -- B:VIUNIX

CREATING, EDITING, AND SAVING FILES

#10 -- ONE -- B:VIUNIX

CREATING, EDIITNG, AND SAVING FILES

This section gives a simple way to create, change and save a file. Other ways to do each of the instructions are covered later in this tutorial or in other texts. The commands covered in this section include:

1. entering or opening a file using the vi command.

2. entering input into a file using the i command.

3. saving the changes and exiting vi with the ZZ command.

4. moving the cursor around within the file using the +, -, <SPACE BAR>, and <BACK SPACE> commands.

5. editing the file using the i, x, and r commands.

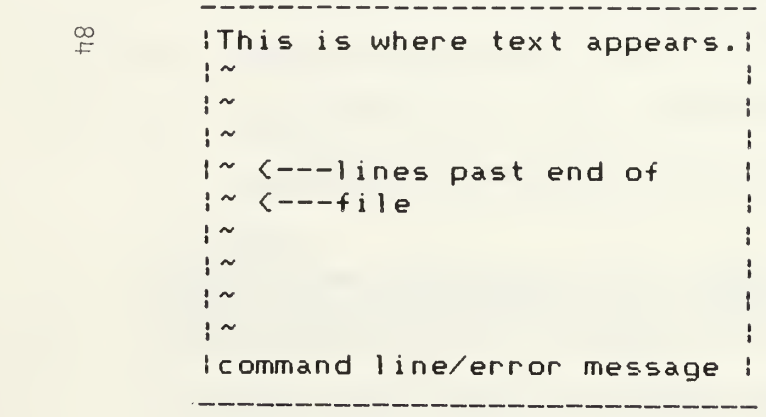
Type <RET> when ready to continue with the tutorial.

#11 -- ONE -- B:VIUNIX

CREATING, EDITING, AND SAVING FILES

A new file is created by the command vi < filename.filetype >. The same command vi < filename.filetype > is used to edit a file that already exists.

Here is a sample of a file. Type vi sample.txt on your other terminal to obtain your own sample file.



To continue with the tutorial,
type NEXT <RET>.
For HELP, type HELP <RET>.

#12 -- ONE -- B:VIUNIX

HELP - CREATING FILES

< vi > is the commmand used to enter or open a file.

< filename > is composed of up to 8 letters and/or numbers and must start with a letter.

< filename.filetype > filetype, which is optional, is also composed of up to 8 letters and/or numbers. Together filename and filetype must not total more than 14 letters and/or numbers.

Type <RET> when ready to return to the tutorial.

#13 -- ONE -- B:VIUNIX

CREATING, EDITING, AND SAVING FILES

Now that you have opened a file, in which to enter input, use the input command i. Everything you type after i will appear on the screen, but i will not. After completing the input, terminate the command with <ESC> which is found in the upper left corner of most terminals.

Go ahead and practise with i on the other terminal. Input several lines of text. Don't worry about mistakes, as you will have the opportunity to correct them later. Remember to terminate the input with <ESC>.

To continue with the tutorial,
type NEXT <RET>.
For HELP, type HELP <RET>.

#13 -- ONE -- B:VIUNIX

CREATING, EDITING, AND SAVING FILES

Now that you have opened a file, in which to enter input, use the input command i. Everything you type after i will appear on the screen, but i will not. After completing the input, terminate the command with <ESC> which is found in the upper left corner of most terminals.

Go ahead and practise with i on the other terminal. Input several lines of text. Don't worry about mistakes, as you will have the opportunity to correct them later. Remember to terminate the input with <ESC>.

To continue with the tutorial,
type NEXT <RET>.
For HELP, type HELP <RET>.

#14 -- ONE -- B:VIUNIX

CREATING, EDITING, AND SAVING FILES

i is also used to insert input into previously created text. This usage also terminates with <ESC>.

Type <RET> when ready to continue.

#15 -- ONE -- B:VIUNIX

HELP - INPUT OR INSERTION OF TEXT

i allows the insertion of input. It can be used at the start of a file or later on in the file. An input session using i terminates with <ESC>.

Type <RET> when ready to return to the tutorial.

#16 -- ONE -- B:VIUNIX

CREATING, EDITING, AND SAVING FILES

After you have terminated your input with <ESC>, save your file with the command ZZ. This command saves the file and exits vi. There will be a line displayed showing the name of the file and the number of lines and the number of characters contained in the file before the UNIX % prompt appears.

Go ahead and use ZZ to save and exit from your file on the other terminal.

To continue with the
tutorial, type NEXT <RET>.
For HELP, type HELP <RET>.

#17 -- ONE -- B:VIUNIX

HELP - SAVING A FILE

ZZ is a single command used to save a file and to exit from vi.

Type <RET> wehn ready to
return to tutorial.

#18 -- ONE -- B:VIUNIX

CREATING, EDITING, AND SAVING FILES

File editing is accomplished by moving the cursor to the
appropriate area and inserting, deleting or replacing a
given character or characters.

Type <RET> when ready to
continue with the tutorial.

#19 -- ONE -- B:VIUNIX

CREATING, EDITING, AND SAVING FILES

In normal mode (when not entering text), the cursor can be moved from the start of one line to the start of another line by (+) (down/forward one line) and (-) (up/backward one line). If used during the insert mode, these characters will appear as + or - .

You can move one character at a time within a line by using <SPACE BAR> (forward one space) and <BACK SPACE> (backward one space).

Go ahead and try the four cursor movement commands on the other terminal.

To continue with the
tutorial, type NEXT <RET>.
For HELP, type HELP <RET>.

#21 -- ONE -- B:VIUNIX

HELP - CURSOR MOVEMENT

Down/forward one line
< + >

Up/backward one line
< - >

Forward one character in a line
< SPACE BAR >

Backward one character in a line
< BACK SPACE >

To return to the tutorial,
type <RET>.

#22 -- ONE -- B:VIUNIX

CREATING, EDITING, AND SAVING FILES

Once the cursor is in position, a character may be inserted, deleted or replaced.

i was previously used to initially enter input. It can also be used to insert a character or characters after the cursor. It terminates with <ESC>.

x deletes the single character indicated by the cursor.

r replaces the single character indicated by the cursor with the single character following r.

Go ahead and try each one of the three editing commands in your file on the other terminal.

To continue with the tutorial,
type NEXT <RET>.
For HELP, type HELP <RET>.

#24 -- ONE -- B:VIUNIX

HELP - EDITING

Inserting one or more characters after the cursor - ends with <ESC>
i

Deleting one character indicated by the cursor
x

Replacing one character indicated by the cursor with one character
r

Type <RET> when ready to
return to the tutorial.

#25 -- ONE -- B:VIUNIX

SUMMARY

Creating a file or entering an old file
vi <filename.filetype>

Saving a file and exiting vi
ZZ

Entering/inserting input (ends with <ESC>)
i

Moving the cursor
< + > forward/down one line
< - > backward/up one line
<BACKSPACE > backward one space
< SPACE BAR > forward one space

Editing a file
x deletes one character
r replaces one character

Type <RET> when ready to return to the menu.

#26 -- ONE -- B:VIUNIX

FILE MANIPULATION

#27 -- ONE -- B:VIUNIX

FILE MANIPULATION

In this section, you will learn how to manipulate files. Specifically, you will learn how to :

1. list the files contained in your directory, using the ls command.

2. remove one or more files, using the rm command.

3. copy a file, using the cp command.

4. rename a file, using the mv command.

5. display one or more files on the terminal, using the cat, more or pr commands.

6. print one or more files on the line printer, using the cat, more or pr commands with the pipeline process.

Type <RET> when ready to continue with the tutorial.

FILE MANIPULATION

The following UNIX commands manipulate one or more files.

ls lists directory of your files.

rm removes files from your directory

cp copies the first file into the second file

mv moves one file into another file (renames)

Type <RET> when ready to
continue with the tutorial.

#29 -- ONE -- B:VIUNIX

FILE MANIPULATION

You should exit from vi at this point. Your directory will contain several simple files (besides the one you created) for use in this section.

Go ahead and try the four commands on the other terminal.

Do not remove all of the sample files as you will need any two or three of them later in this section.

To see the previous screen
again, type BACK <RET>.
To continue with the
tutorial, type NEXT <RET>.
For HELP, type HELP <RET>.

#30 -- ONE -- B:VIUNIX

HELP - FILE MANIPULATION

ls lists your file directory

rm removes files from your directory

cp copies the first file into the second file

mv moves one file into another file (renames)

Type <RET> when ready to
return to the tutorial.

#31 -- ONE -- B:VIUNIX

FILE MANIPULATION - TERMINAL DISPLAY

In this section, you will learn how to display one or more files at terminal in formatted or unformatted form.

pr <filenames> formats one or more files by adding date, time and page numbers and separates the files.

cat <filenames> concatenates one or more files and displays them at the terminal without pause. The files are unformatted and unseparated.

more <filename> displays one file screen by screen. It continues from one screen to the next by hitting the <SPACE BAR>.

Go ahead and try the three terminal display commands using the files in your directory on the other terminal.

To continue with the tutorial, type NEXT <RET>.
For HELP, type HELP <RET>.

#33 -- ONE -- B:VIUNIX

HELP - TERMINAL DISPLAY

pr < filenames > formats the files and displays (prints) them separately.

cat < filenames > concatenates files and displays them at the terminal
without pause. Files are unformatted and not separated.

more < filename > displays a file one screenful at a time.

Type <RET> when ready to
return to the tutorial.

#34 -- ONE -- B:VIUNIX

FILE MANIPULATION - HARD COPY

There is no one command to direct output to the line printer. UNIX uses a pipeline process, whereby the output from one command becomes the input to the next command. The commands for the display/print portion are the same for the line printer as they were for the terminal.

e.g. cat PadsIn PadsOut | lpr
pr page.txt section.txt | lpr

| is the pipeline character and is found to the right of the keyboard near <RET>. As shown in the sample above, the commands will only print out on the line printer. As before, cat prints unformatted and does not separate files, while pr both formats and separates.

Type <RET> when ready to
continue with the tutorial.

#35 -- ONE -- B:VIUNIX

FILE MANIPULATION - HARD COPY

Go ahead and try the two hard copy commands on the other terminal with the files in your directory.

The line printer is located in S-513 (the room behind you). Make sure the line printer is turned on before you issue the print commands.

To see the previous screen
again, type BACK <RET>.
To continue with the
tutorial, type NEXT <RET>.
For HELP, type HELP <RET>.

#36 -- ONE -- B:VIUNIX

HELP - HARD COPY

```
cat < filenames > | lpr

pr < filenames > | lpr
```

Both of these commands will print one or more files on the line printer. The sole difference is that cat does not format or separate the files and pr does.

Type <RET> when ready to return to the tutorial.

#37 -- ONE -- B:VIUNIX

FILE MANIPULATION - SUMMARY

ls lists file directory.

rm < filenames > removes files from directory.

cp < filename1 filename2 > copies first file into second file.

mv < filename1 filename2 > moves first file into second file.

pr < filenames > formats and prints files at terminal

cat < filenames > prints files at terminal.

more < filename > prints file at terminal by screenfuls.

pr < filenames > | lpr formats and prints files on line printer.

cat < filenames > | lpr prints files on line printer.

Type <RET> when ready
to return to the menu.

#38 -- ONE -- B:VIUNIX

MORE ON CURSOR MOVEMENT

#39 -- ONE -- B:VIUNIX

MORE ON CURSOR MOVEMENT

The cursor movement commands already learned were all small scale.
The commands in this section are large scale.

^F moves the cursor forward one screenful (about 20 lines).

^B moves the cursor backward one screenful (about 20 lines).

^D moves the cursor forward 1/2 screenful (about 11 lines).

^U moves the cursor backward 1/2 screenful (about 11 lines).

nG moves the cursor to line n (default is last line).

Go ahead and try the cursor movement commands on one of the longer files in
your directory on the other terminal.

To continue with the
tutorial, type NEXT <RET>.
For HELP, type HELP <RET>.

#41 -- ONE -- B:VIUNIX

HELP - CURSOR MOVEMENT

Commands that scroll up or move backward
        ^B goes backward 1 screenful (about 20 lines).
        ^U goes backward 1/2 screenful (about 11 lines).

Commands that scroll down or move forward
        ^F goes forward 1 screenful (about 20 lines).
        ^D goes forward 1/2 screenful (about 11 lines).

Command that moves to a specific line
        nG goes to line n. G is the default and goes to the last line.

111

Type <RET> when ready to
return to the tutorial.

#42 -- ONE -- B:VIUNIX

SUMMARY - MORE ON CURSOR MOVEMENT

Commands that scroll down or move forward
        ^F goes forward 1 screenful (about 20 lines).
        ^B goes forward 1/2 screenful (about 11 lines).

Commands that scroll up or move backward
        ^B goes backward 1 screenful (about 20 lines).
        ^U goes backward 1/2 screenful (about 11 lines).

#43 -- ONE -- B:VIUNIX

MORE ON EDITING FILES

#44 -- ONE -- B:VIUNIX

MORE ON EDITING FILES

In this section you will learn more about editing files. Specifically, you will learn additional methods to:

1. append or insert input using the a, A, i, and I commands.

2. open lines in a file, using the o and O commands.

3. delete a character or characters, using the dd, D, x, and X commands.

4. replace a character or characters, using the r and R commands.

5. undo changes, using the u and U commands.

6. join lines, using the J command.

Type <RET> when ready to
continue with the tutorial.

#45 -- ONE -- B:VIUNIX

MORE ON EDITING FILES - ENTERING INPUT

The following commands provide additional ways to enter input. Commands given previously are shown here for contrast.

a appends input after (to right of) cursor.
A appends input at end of current line.

i inserts input before (to left of) cursor.
I inserts input at beginning of current line.

o opens a line above the current line.
O opens a line below the current line.

The above commands all terminate with <ESC>. They can be used either to enter input initially or in an already existing file.

Go ahead and try the 6 input entering commands with one of your files on the other terminal. Note the importance of uppercase vs. lowercase letters with regard to the command's result.

To continue with the tutorial, type NEXT <RET>.
For HELP, type HELP <RET>.

#47 -- ONE -- B:VIUNIX

HELP - ENTERING INPUT

Appending input
        a appends after (to right of cursor).
        A appends at end of current line.

Inserting input
        i inserts before (to left of cursor).
        I inserts at beginning of current line.

Opening a line
        o opens above current line.
        O opens below current line.

Type <RET> when ready to
return to the tutorial.

#48 -- ONE -- B:VIUNIX

MORE ON EDITING FILES - CHANGING TEXT

The following commands are additional ways to change input. Some have been previously given and are shown here for contrast.

x deletes the single character indicated by the cursor.
X deletes the single character before (to the left of) the cursor.

dd deletes the remainder of the line and joins the next line on to it (wrap-around).
D deletes the remaining characters on the line (creates blank space).

r replaces the single character covered by the cursor with a single character.
R replaces text until terminated with <ESC>.

Go ahead and try the 6 commands for changing text on one of your files on the other terminal. Note the difference between dd and D.

To continue with the tutorial, type NEXT <RET>.
For HELP, type HELP <RET>.

#50 -- ONE -- B:VIUNIX

HELP - CHANGING TEXT

Deleting a character
        x deletes the character indicated by the cursor.
        X deletes the character before the cursor.

Deleting a line
        dd deletes the rest of the line and wraps the next line around.
        D blanks the rest of the line.

Replacing a character or characters
        r replaces the character indicated by the cursor with a single
          character.
        R replaces a string of text character for character until
          terminated by <ESC>.

Type <RET> when ready to
return to the tutorial.

#51 -- ONE -- B:VIUNIX

MORE ON EDITING FILES - REPAIR

These commands perform "repair" functions.

u undoes the last change.
U restores the current line.

J joins lines.

Go ahead and create some mistakes in one of your files on the other terminal. Then use the 3 repair commands to fix them.

To continue with the
tutorial, type NEXT <RET>.
For HELP, type HELP <RET>.

#53 -- ONE -- B:VIUNIX

HELP - REPAIR

Undoing changes
        u undoes last change.
        U restores last line.

Joining lines
        J joins lines.

Type <RET> when ready to
   continue with the tutorial.

#54 -- ONE -- B:VIUNIX

MORE ON EDITING - SUMMARY

a	appends after cursor and terminates with <ESC>.
A	appends at end of current line and terminates with <ESC>.
i	inserts before cursor and terminates with <ESC>.
I	inserts at start of current line and terminates with <ESC>.
o	opens line above current line and ends with <ESC>.
O	opens line below current line and terminates with <ESC>.
x	deletes a character indicated by the cursor.
X	deletes a character before the cursor.
dd	deletes the rest of the current line and wraps the next line around.
D	blanks the rest of the current line.
r	replaces one characer with another.
R	replaces a string of text until terminated with <ESC>.
u	undoes last change.
U	restores current line.
J	joins lines.

Type <RET> when ready
to return to the menu.

#55 -- ONE -- B:VIUNIX

MORE ON SAVING FILES

#56 -- ONE -- B:VIUNIX

MORE ON SAVING FILES

The following commands provide alternate methods for saving a fil .
The colon : commands will appear at the bottom of the screen on the
command line.

:w writes back (saves) changes and stays in vi.

:wq writes back (saves) changes and exits vi. Same as ZZ

:q quits vi only if no changes made to file.

:q! quits vi and destroys (does NOT save) changes.

^Z aborts the vi session, does not save changes and creates a stopped
    job.

Go ahead and try the 6 saving commands on one of the files on the
other terminal.

To continue with the
tutorial, type NEXT <RET>.
For HELP, type HELP <RET>.

#59 -- ONE -- B:VIUNIX

SAVING FILES - SUMMARY

Writing back (saving) changes

```
        :w  saves and stays in vi.
        :wq saves and exits vi.
        ZZ  saves and exits vi.
```

Not saving changes

```
        :q  if no changes made, exits vi.
        :q! ignores changes made and exits vi.
        ^Z  ignores changes made, aborts the session and creates a
            stopped job.
```


Type <RET> when ready
to return to the menu.

#60 -- ONE -- B:VIUNIX

OTHER TUTORIALS AND ON-LINE HELP

#61 -- ONE -- B:VIUNIX

OTHER TUTORIALS AND ON-LINE HELP - MANUALS AND HARD-COPY HELP

The following manuals or pamphlets are available in S-511 or S-525A.

"An Introduction to Display Editing with Vi" by William Joy.

"UNIX for Beginners - Second Edition" by Brian W. Kerighan.

"VI QUICK REFERENCE" (outstanding 1 page summary of vi, ex commands).

man <name of command> | lpr will provide a hard copy of the UNIX manual's explanation of the command. There is a copy of the whole manual in S-511.

Type <RET> when ready to continue with the tutorial.

#62 -- ONE -- B:VIUNIX

OTHER TUTORIALS AND ON-LINE HELP - ON-LINE HELP

vi has no on-line help. If something goes wrong in vi, the error statements appearing on the last line of the screen will be cryptic.

man <name of command> is the on-line help for UNIX. Where applicable, each command is set up in a more format.

Type <RET> when ready to continue with the tutorial.

#63 -- ONE -- B:VIUNIX

OTHER TUTORIALS AND ON-LINE HELP - TUTORIALS

vi.tutorial is a very lengthy tutorial that covers vi in exhaustive detail. Use it by typing vi vi.tutorial and following the instructions.

learn has a series of tutorials on vi, ex (another editor), C (a command language) and on files. They can be long.

Type <RET> when ready
to return to the menu.

#64 -- ONE -- B:VIUNIX

THE END

THANK YOU FOR USING THE UNIX AND VI TUTORIAL.

IF YOU HAVE ANY COMMENTS OR SUGGESTIONS, PLEASE
ENTER THEM AT THIS TIME. TERMINATE YOUR INPUT
WITH <RET>.

IF YOU DO NOT HAVE ANY INPUT, HIT <RET>.

TERMINATE THE TUTORIAL BY ENTERING $.

#66 -- ONE -- B:VIUNIX

MENU

PROMPT INDICATES LAST SECTION COMPLETED.

A. TUTORIAL CONVENTIONS

B. UNIX CONVENTIONS

C. CREATING, EDITING, AND SAVING FILES

D. FILE MANIPULATION

E. MORE ON CURSOR MOVEMENT

F. MORE ON EDITING FILES

G. MORE ON SAVING FILES

H. OTHER TUTORIALS AND ON-LINE HELP

I. EXIT TUTORIAL

#67 -- ONE -- B:VIUNIX

MENU

A. TUTORIAL CONVENTIONS

B. UNIX CONVENTIONS

C. CREATING, EDITING, AND SAVING FILES

D. FILE MANIPULATION

E. MORE ON CURSOR MOVEMENT

F. MORE ON EDITING FILES

G. MORE ON SAVING FILES

H. OTHER TUTORIALS AND ON-LINE HELP

I. EXIT TUTORIAL

PROMPT INDICATES LAST SECTION COMPLETED.

#68 -- ONE -- B:VIUNIX

MENU

A. TUTORIAL CONVENTIONS

B. UNIX CONVENTIONS

C. CREATING, EDITING, AND SAVING FILES

D. FILE MANIPULATION

E. MORE ON CURSOR MOVEMENT

F. MORE ON EDITING FILES

G. MORE ON SAVING FILES

H. OTHER TUTORIALS AND ON-LINE HELP

I. EXIT TUTORIAL

PROMPT INDICATES LAST SECTION COMPLETED.

REPRODUCED AT GOVERNMENT EXPE

#69 -- ONE -- B:VIUNIX

MENU

A. TUTORIAL CONVENTIONS

B. UNIX CONVENTIONS

C. CREATING, EDITING, AND SAVING FILES

D. FILE MANIPULATION

E. MORE ON CURSOR MOVEMENT

F. MORE ON EDITING FILES

G. MORE ON SAVING FILES

H. OTHER TUTORIALS AND ON-LINE HELP

I. EXIT TUTORIAL

PROMPT INDICATES LAST SECTION COMPLETED.

#76 -- ONE -- B:VIUNIX

TO LOGIN/LOGOUT TO UNIX (ON ANY UNIX TERMINAL)

LOGIN PROCEDURE: 4.2 BSN UNIX (nps-cs)

login: (Type your last name or the class name if it is a class account.)

PASSWORD: (Type your password or the class password if it is a class account.)

TERM = (vt100) (Just hit <RET>.)

(If you are working under a class account, there will be a request for your last name at this point.)

LOGOUT PROCEDURE: (Type logout. If the comment is made, "There are stopped jobs.", type logout again.)

APPENDIX B

SELECTED FRAMES FROM THE VLSI TUTORIAL

Figures B.1, B.2, B.3, B.4, and B.5 are intended to illustrate some of the graphics capabilities of the VCIS. These frames do not constitute the entire tutorial nor do they show all the graphics techniques available under VCIS.

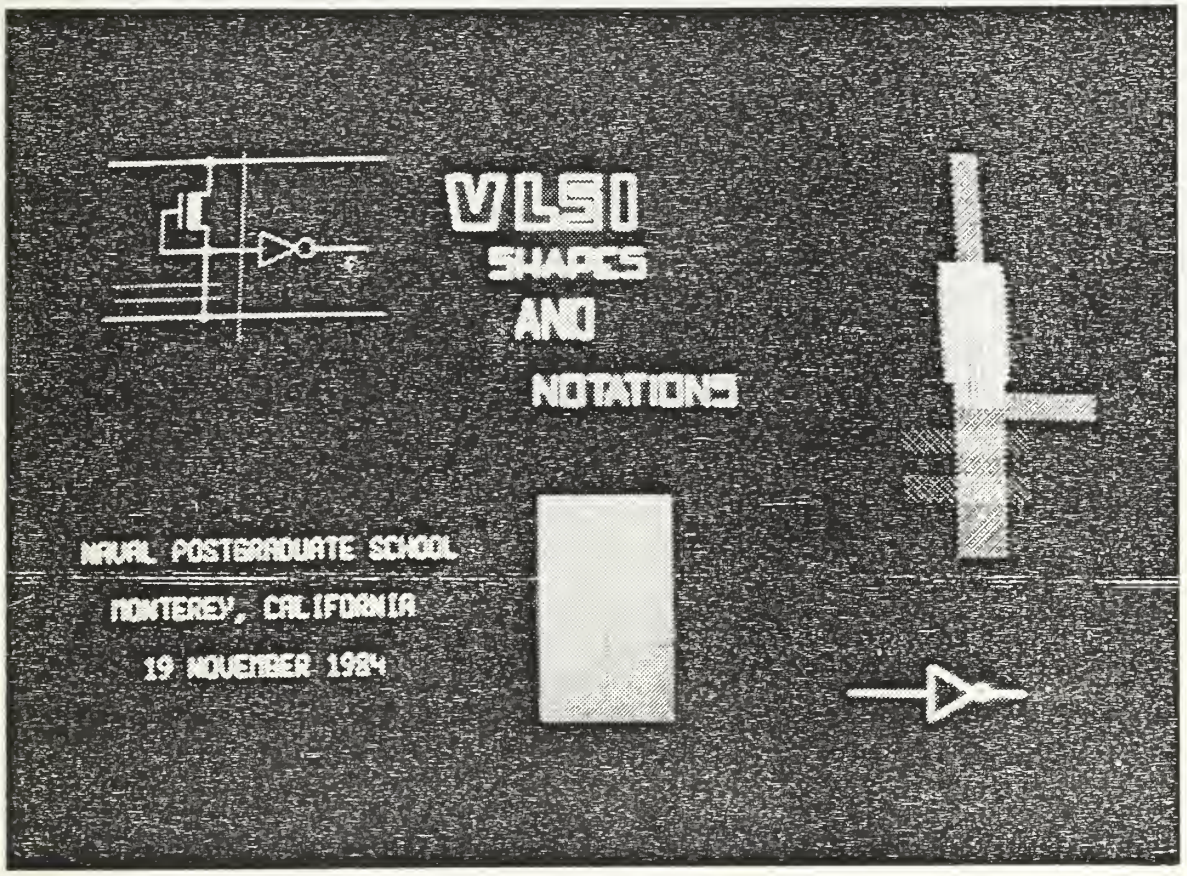


Figure B.1 VLSI Tutorial Logo

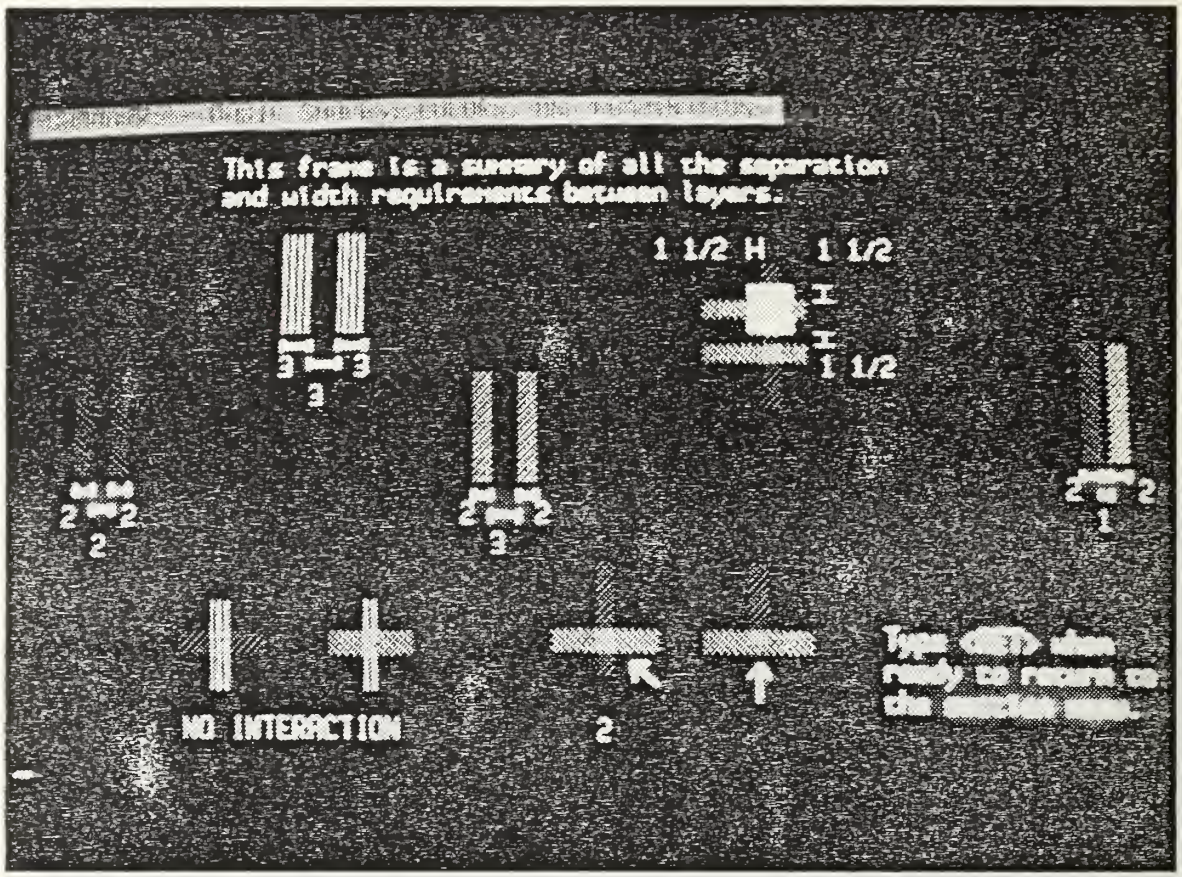


Figure B.2 Summary of Basic Shapes, Colors, and Combinations

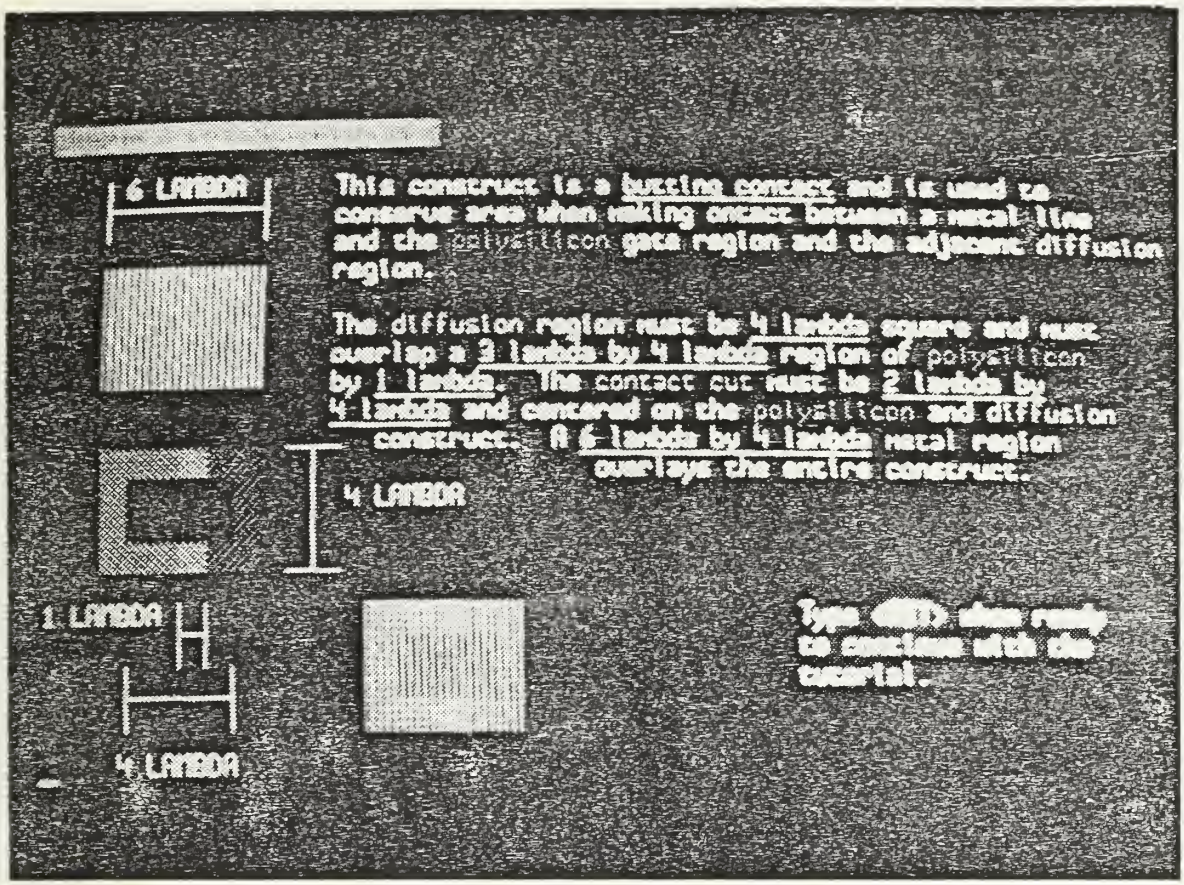


Figure B.3 Butting Contact

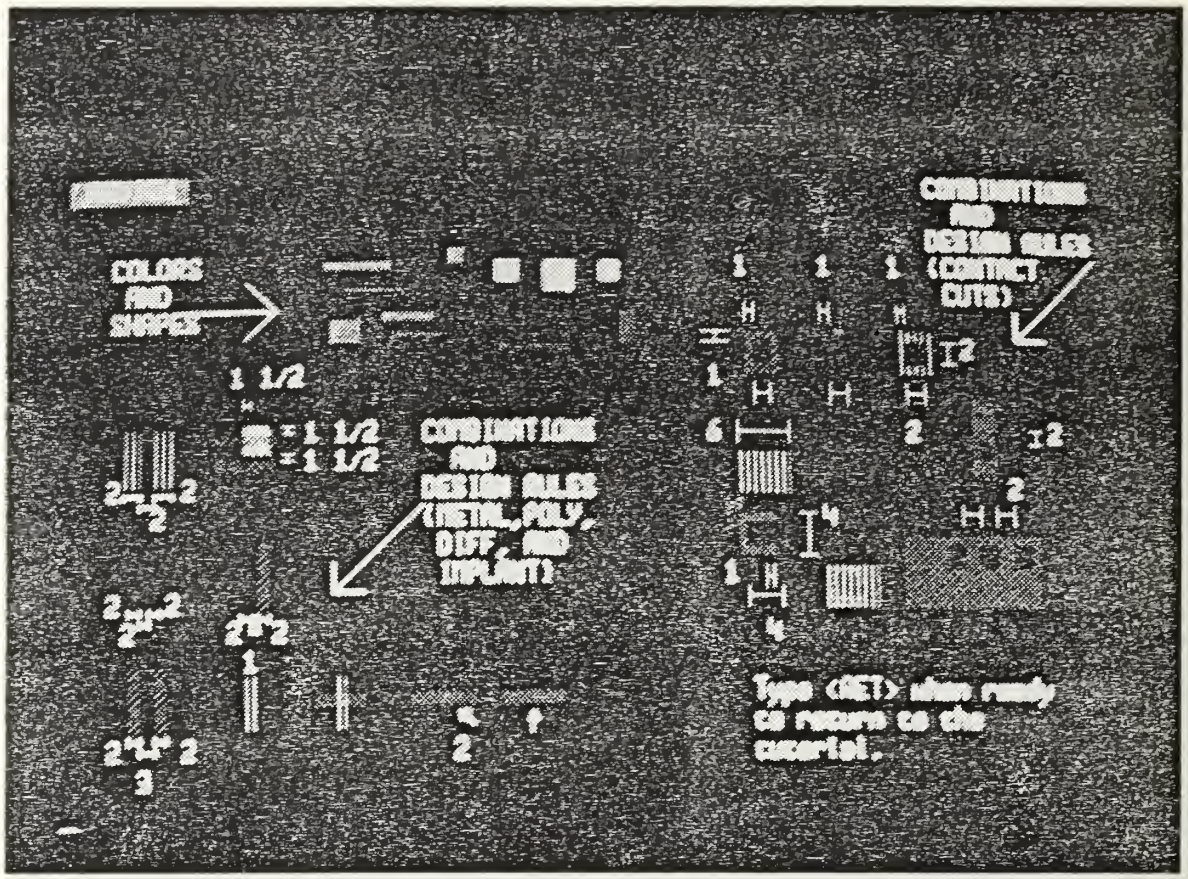


Figure B.4 Review

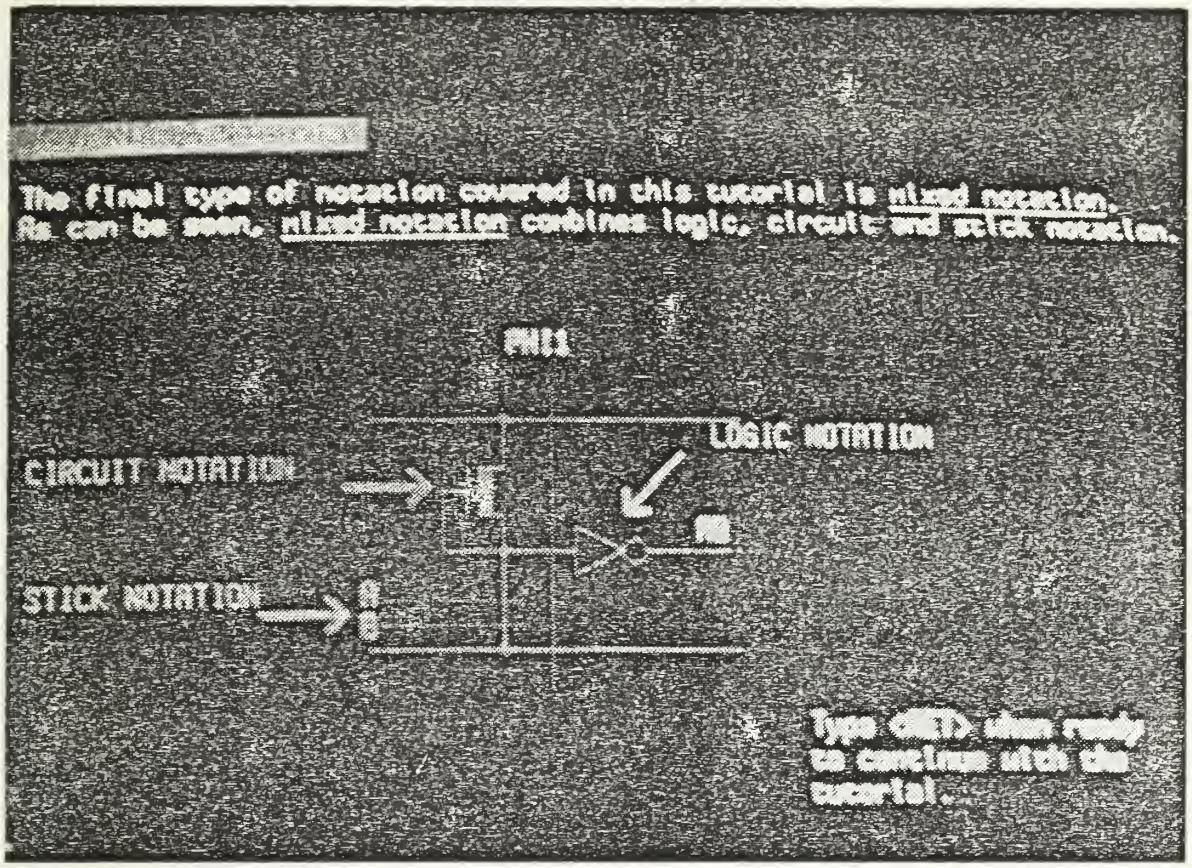


Figure B.5 Mixed Notation

APPENDIX C

SYSTEM REQUIREMENTS AND LOGON PROCEDURE FOR VCIS

1. Diskettes

The IBM PC with VCIS uses double sided double density 5 1/4 inch floppy diskettes. TEXTEDIT, GRAFEDIT, and BUILDER each should have their own diskette due to their size. INTERP and MANAGER can reside on the same diskette. CHEDIT, LISTFRAM, and LINKER can also reside on the same floppy.

2. Logon Procedure

A. Turn on the disk drive. The orange power switch is located to the right and near the back of the disk drive unit. Then turn on the monitor.

B. The booting disk should be inserted label up in the leftmost or upper disk drive, also known as drive 0 or drive A. Where the location of drive 0 is, depends on the configuration of the system.

C. If the booting disk is not inserted before the IBM PC is turned on, depressing <CONTROL>, <DEL>, and <ALT> simultaneously will boot the system.

D. Rarely, booting the system does not clear the screen or produce the logon text. At this point, the user must turn the disk drive unit off for a few minutes and then re-boot.

E. A successful booting will ask for the current date and the current time. If desired, they must be entered as in the sample on the screen. If no data or time needs to be attached to the session, typing <RET> twice cycles the user past this block.

F. The booting or system disk that comes with the Microsoft Dos 2.0 will not work with VCIS. VCIS restructures the keyboard in a certain manner, which is not done by the standard booting diskette.

3. Logging Off

Provided that either the tutorial authoring session is complete, or that the tutorial execution is over, and the drive prompt has been displayed, the user may logoff simply by removing all diskettes and turning off the power switches previously mentioned. Removing the disks while the disk drive is moving is an emergency measure only and should be avoided if at all possible.

4. Executing The VIUNIX Tutorial

A. Boot the system using the disk labelled "Tecmar booting disk". The prompt is "A>".
B. Switch to b drive by typing "b:<RET>".
C. Insert disk labelled "Viunix tutorial" or "VLSI Shapes tutorial" label up into right or lower drive. Type "viunix<RET>" or "shapes<RET>", depending on which tutorial is being executed. There will be an initial flurry of colors/shapes. The tutorial is self-explanatory after this.
D. If the user needs to leave the tutorial and is at a decision point (input is required the user), type <ESC>. The prompt is "Resume Stop ". Type only "S" (or "R" if the user has changed his/her mind). Do not hit return - only the letter is necessary.
E. Generally, if the user types in something very wrong, nothing should happen. The tutorial will wait at that screen until a correct response is entered. If the user inputs a misspelling that is at least 75% correct and sufficiently different from another correct response, the tutorial continues on what is assumed to be the correct path.

For example, if the possible responses are "NEXT HELP", "NEXX" or "MEXT" will work, but "HEXT" will not as that could be either "NEXT" or "HELP". A single character response must be 100% correct, of course.

F. Logoff the PC using the procedure described previously.

APPENDIX D

SURVEY ON TUTORIAL

LIESEL MUTH
TUTORIAL SURVEY
12 OCTOBER 1984

1. EXPLANATION.

THIS IS ONE IN A SERIES OF TUTORIALS ON VLSI DESIGN AND ASSOCIATED REQUIREMENTS. I WOULD LIKE YOUR SUGGESTIONS/FEELINGS/COMMENTS ON A VARIETY OF AREAS.

2. TIMED FRAMES.

THERE ARE SEVERAL FRAMES (USUALLY AT THE START OF A TUTORIAL) THAT ARE VISIBLE FOR A SET AMOUNT OF TIME. PLEASE COMMENT ON THE AMOUNT OF TIME - IS IT ENOUGH OR IS IT TOO MUCH TIME? WOULD IT HAVE BEEN BETTER TO HAVE BEEN ALLOWED TO CONTINUE WHEN READY RATHER THAN HAVING A TIMED FRAME?

2. MENU.

IS THE USE OF THE MENU CLEAR? WOULD ANOTHER METHOD HAVE BEEN BETTER OR LESS CONFUSING? IF YES, PLEASE EXPLAIN WHERE THE CONFUSION EXISTS AND WHAT COULD BE DONE TO IMPROVE IT.

3. USE OF COLORS IN TEXT.

WERE ANY OF THE COLORS USED IN THE TEXT IRRITATING OR UNNECESSARY? IF YES, PLEASE IDEDTIFY THE FRAME(S) AND EXPLAIN WHY.

4. GRAPHICS.

AS NOT ALL THE TUTORIALS USE GRAPHICS, PLEASE SKIP THIS SECTION IF IT DOES NOT APPLY.

WERE ANY OF THE GRAPHICS CONFUSING? IF YES, PLEASE IDENTIFY THE FRAME(S) AND EXPLAIN WHY.

WERE THERE ANY FRAMES WHICH CONTAINED TOO MANY ILLUSTRATIONS? IF YES, PLEASE IDENTIFY THE FRAME(S) AND WHICH ILLUSTRATIONS SHOULD BE REMOVED.

5. SUMMARY FRAMES.

ARE THE SUMMARY FRAMES ADEQUATE? SHOULD THERE BE MORE OR LESS INFORMATION ON THEM? PLEASE IDENTIFY THE FRAME(S) AND STATE HOW THEY COULD BE IMPROVED.

6. DUAL SCREEN PRESENTATION.

DUAL SCREEN PRESENTATION IS NOT USED ON ALL TUTORIALS. PLEASE SKIP THIS SECTION IF IT DOES NOT APPLY.

IS THE DUAL SCREEN PRESENTATION EFFECTIVE? WOULD IT HAVE BEEN BETTER TO HAVE USED ONLY ONE TERMINAL?

7. STIPPLE PLOTS.

STIPPLE PLOTS ARE NOT AVAILABLE ON ALL TUTORIALS. PLEASE SKIP THIS SECTION IF IT DOES NOT APPLY.

DID YOU FIND THE STIPPLE PLOTS USEFUL/EFFECTIVE? WOULD YOU HAVE WANTED MORE OR FEWER PLOTS?

8. COMMENTS

ARE THERE ANY OTHER COMMENTS? ARE THERE ANY AREAS TO WHICH MATERIAL COULD BE ADDED/DELETED? ARE THERE ANY AREAS WHICH NEED IMPROVEMENT? PLEASE SPECIFY SECTION(S) AND FRAMES WHERE POSSIBLE.

MANY THANKS FOR TAKING THE TIME TO DO THIS.

LIST OF REFERENCES

1. Knapp, Barbara H., and Richard C. Brandt, The University of Utah Video-computer Courseware Implementation System User's Guide, Version II.0d, 1983.

2. Knapp, Barbara H., and Richard C. Brandt, The University of Utah Video-computer Courseware Implementation System User's Guide, Version III.0a, Draft, August, 1984.

3. Carver, Mead, and Lynn Conway, Introduction to VLSI Systems, Addison-Wesley, Reading, MS, 1980.

BIBLIOGRAPHY

Birren, Faber, Color and Human Response, Van Nostrand Reinold Co., NY, 1978.

Birren, Faber, Color Psychology and Color Therapy, Van Nostrand Reinhold Co., New York, 1978.

Dean,Christopher, and Whitlock,Quentin, A Handbook of Computer Based Training, Kogan Page, London, 1983.

DeCecco, John P., Educational Technology Readings In Programmed Instruction, Holt, Rinehart, & Winston, New York, 1964.

Gerard, R. W., Computers and Education - A Workshop, McGraw-Hill Book Corp., New York, 1967.

Hartley, James, Trueman, Mark, Burnhill, Peter, "Some Observations On Producing and Measuring Readable Writing," Programmed Learning and Educational Technology, Vol. 17, No. 3, Aug. 1980.

Joy, William, An Introduction to Display Editing with Vi, University of Berkeley.

Kaplan, R. E., and Walts, D. L., A Design for an Interactive Videodisc Training Program for the Sun Workstation, MS Thesis, Naval Postgraduate School, Monterey, CA, March, 1984.

Kernighan, Brian W., UNIX For Beginners, 2nd ed., Bell Laboratories, 1978.

UW/NW VLSI Consortium, UNIXquick - Getting a quick start on UNIX, 1984.

INITIAL DISTRIBUTION LIST

No.		Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943	2
3.	Barbara H. Knapp Dept. of Computer Science University of Utah Salt Lake City, Utah	2
4.	Atsuko Keyiu, Code 62Le Dept. of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943	1
5.	Dr. M. Woods, Code 001 Dean of Continuing Education Naval Postgraduate School Monterey, CA 93943	1
6.	Dr. D. E. Kirk, Code 62Ki Dept. of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943	2
7.	LT Liesel R. Muth HQ AFSOUTH Box 139 FPO NY, NY 09524	2

Thesis
M9867
c.1

Muth
VLSI tutorials through the Video-computer Courseware Implementation System.

Thesis
M9867
c.1

Muth
VLSI tutorials through the Video-computer Courseware Implementation System.

thesM9867
VLSI tutorials through the Video-compute
3 2768 000 61060 4
DUDLEY KNOX LIBRARY